Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Irányítástechnika és Informatika Tanszék

Számítógépes Látórendszerek

Jegyzet

Szemenyei Márton

2023. szeptember 20.

Tartalomjegyzék

I. rész

Hagyományos Látás

1. fejezet

Bevezetés a Számítógépes Látásba

1.1. Mi az a számítógépes látás?

A különféle számítógépes látórendszerek az élet egyre több területén váltak elterjedtté. Ez a rendelkezésre álló számítási teljesítmény és az eszközök rohamos gyarapodásának, valamint a rendszerekben használt algoritmusok jelentős fejlődésének is köszönhető. Ezen algoritmusoknak számos felhasználási területe létezik: a robotikától és az automatizálástól kezdve a virtuális- és kiterjesztettvalóság-rendszereken keresztül egészen a szórakoztatóiparig. Jelentős továbbá e módszerek közszolgálati területeken történő alkalmazásának lehetősége is.

A számítógépes látás kutatási területének az a célja, hogy algoritmikus eszközök segítségével képekből vagy képsorozatokból egy másik döntéshozó alkalmazás vagy személy számára releváns információkat nyerjünk ki. Az elvégzendő feladat legnagyobb nehézsége, hogy akár egyetlen kép is milliós nagyságrendű adatból (képpontból) áll, amelyek ráadásul ennél is számos nagyságrenddel több konfigurációban alkothatnak képeket. Ilyen adatmennyiség feldolgozásához hatékony algoritmusokra és nagy teljesítményű eszközökre van szükség.

A terület további nehézsége, hogy habár az ember képes egy látott kép alapján számos létfontosságú információt meghatározni (sőt, a szem az ember legfontosabb érzékszerve), e feldolgozás nagy részét tudat alatt végezzük, így nem tudjuk ezeket a képességeket könnyen egzakt algoritmusra váltani. Ezt a problémát tovább nehezíti, hogy feltehetően számos látásalapú döntésünkhöz felhasználunk olyan információkat is, amelyeket egy másik érzékszervünk segítségével nyertünk. A fenti problémák miatt a számítógépes látás területén gyakorta alkalmazunk heurisztikákra, illetve gépi tanulásra, matematikai optimalizálásra épülő eljárásokat, amelyeknek a minden eshetőségre való helyes működését nem tudjuk garantálni.

Ezeket az algoritmusokat számos különböző módon csoportosíthatjuk. Ezek közül az egyik legelterjedtebb az eljárások célja (kimenete) alapján történő csoportosítás. Itt megkülönböztetjük a képfeldolgozás (angolul: image processing), valamint a számítógépes látás (angolul: computer vision) algoritmusait. A képfeldolgozás esetén a célunk az, hogy az algoritmus eredményeként egy olyan új képet kapjunk, amely valamilyen szempontból számunkra előnyösebb, mint az eredeti kép volt. Ezek az eljárások gyakorta hivatottak a képek további feldolgozását vagy éppen az ember általi láthatóságát segíteni.

A számítógépes látás algoritmusai ezzel szemben kimenetükön nem egy újabb képet, hanem valamilyen, a bemenetből kinyert, magasabb absztrakciós szinten létező információt szolgáltatnak. Ezenfelül külön meg szoktuk különböztetni azt az esetet, amikor ezeket az eljárásokat egy beágyazott rendszerben (gép, autó, robot, telefon stb.) használjuk, amely esetben gépi látásról (angolul: machine vision) beszélünk. Elterjedt kifejezés továbbá a videoanalitika, amely esetben az algoritmus bemenete egy képsorozat.

A számítógépes látáson belül gyakorta meg szoktuk különböztetni azokat a megoldásokat, amelyek a gépi tanulás (angolul: machine learning) algoritmusait használják a működésük során, és

ezt a területet tanuló látásként (angolul: learning vision) is hívjuk. Ezeken belül külön figyelmet érdemelnek azok a megoldások, amelyek az elmúlt néhány évben rendkívül népszerűvé vált mélytanulás (angolul: deep learning) megoldásait alkalmazzák. A terület többi módszerét – sokszínűségük ellenére – hagyományos látásnak nevezzük.

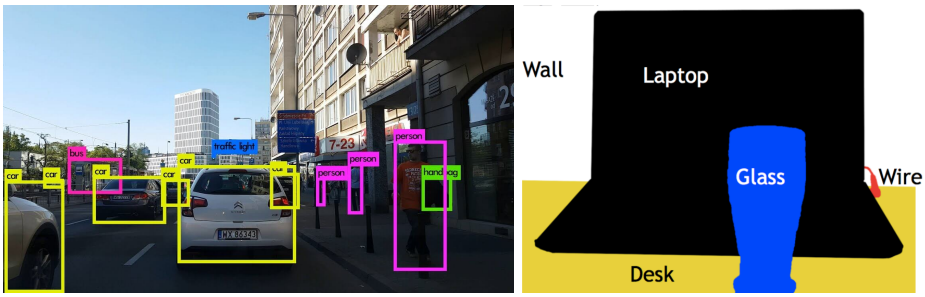
A hagyományos látás megoldásai jellemzően négy fontos lépésből épülnek fel, melyek közül az első a feldolgozandó képek, vagy képsorozatok készítése. Ezt jellemzően egy képjavító lépés követi, melynek során az esetleges képi hibák és zajok hatását kívánjuk minimalizálni. Ebben a lépésben egyéb a feldolgozást segítő konverziókat is végezhetünk. A harmadik lépés a képből az adott feladat számára hasznos jellemzők kinyerése, melyek segítségével a probléma megoldható. Az utolsó lépésben az algoritmus a kinyert jellemzők és egyéb adatok segítségével meghatározza a végső kimenetet - ezt a lépést hívjuk általánosságban döntéshozásnak.

1.1. ábra. *A hagyományos látás algoritmikus lépései.*

1.1.1. Alapvető feladatok és nehézségek

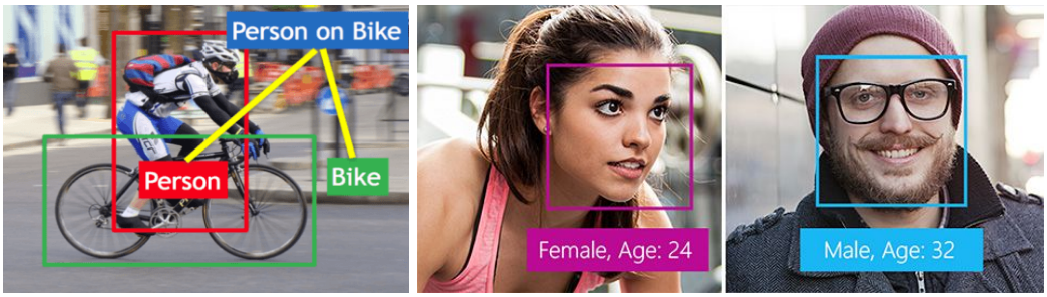
A számítógépes látás területének alapvető célja magas szintű információk kinyerése a képekből. Ennek legegyszerűbb formája az osztályozás feladata, vagyis amikor egy képhez egyetlen címkét rendelünk, amely a képen található objektum kategóriáját kódolja. Bizonyos esetekben a címke mellé már egy az adott objektumot körbefogó téglalapot is rendelünk, ebben az esetben lokalizációról beszélhetünk. A valóságban előforduló képeken azonban több fajta objektum több példányban is előfordulhat, ekkor minden egyes releváns objektum felismerésére és lokalizálására szükség van. Ezt a feladatot nevezzük detektálásnak.

Előfordulhat, hogy az objektumok helyzetén felül annak formájáról és pózáról is szükséges információkat gyűjtenünk, amelyre az objektumokat befoglaló téglalap nem megfelelő. Ekkor célszerű lehet a kép minden egyes pixelét külön osztályozni, így egy olyan maszk képet előállítani, ahol az egyes pixelek értéke annak az objektumnak az osztályát kódolja, amihez az adott képpont tartozik. Ezt a feladatot szemantikus szegmentálásnak nevezzük. Ennek a feladatnak egyik hiányossága, hogy az egymással érintkező azonos osztályú objektumok összeolvadnak, amit elkerülendő a pixeleknek külön osztály és objektumcímkét is rendelhetünk, így eljutva az objektum szegmentálás feladatáig.



1.2. ábra. *A detektálás és a szegmentálás feladata.*

Az objektumok minél pontosabb és magasabb szintű észlelésénél nem kell természetesen megállnunk, megpróbálhatjuk a képből az egyes objektumok tulajdonságait (pl. emberek neme, kora, hangulata) és azok közti kapcsolatokat, összefüggéseket (pl. tartalmazás, geometriai kapcsolatok, tevékenységek) kinyerni, és rendszerbe szedni. Ilyen összetett információk alapján megalapozott döntéseket lehetünk képesek hozni vizuális információk alapján. Ezt a feladatot hívják jelenet értelmezésnek.



1.3. ábra. *Objektum relációk (balra), és kor regresszióval kombinált arcdetektálás (jobbra).*

Amennyiben magas szintű, szemantikus információt szeretnénk a képből kinyerni, számos nehézséggel kell szembenéznünk. Ezek közül az első, hogy ugyanannak az objektumnak a képe különböző megvilágítások miatt jelentős változásokon mehet keresztül, így az objektumot adó pixelek numerikus értéke megközelítőleg sem fog megegyezni. Hasonló nehézségeket eredményez az elforgatás, skálázás és a perspektív torzítás, amelyek ugyanarról az objektumról készített felvételeken ugyanúgy számottevő változásokat okoznak. További problémákkal járhat, hogy egyes objektumok képesek deformálódni, amely szintúgy megváltoztatja a leíró jellemzők értékét. Valódi képeknél szintén gyakori, hogy az objektumok egy jelentős része takarásban van, így a felismerést csak egy részleges kép alapján tudjuk elvégezni.

Az osztályozási és detektálási feladat során azonban nem egyetlen konkrét objektumot, hanem egy szemantikus osztályt szeretnénk felismerni. Egy osztályba számtalan különböző objektum tartozhat, amelyek között (az adott osztálytól függően) jelentős eltérések lehetnek. Sőt, a valós világban gyakoriak az olyan osztályok, amelynek egyes példányai egyáltalán nem mutatnak vizuális hasonlóságot, az azonos osztályba való tartozásukat pedig valamilyen fizikai vagy funkcióbeli hasonlóság alapján tudnánk eldönteni (gondoljunk például különböző kialakítású székekre). Ezt a problémát osztályon belüli variációnak nevezzük, és a szemantikus osztályozása egyik legnagyobb nehézsége.

A fent említett nehézségekhez hozzáadódik még az úgynevezett szemantikus gát. A szemantikus gát fogalma foglalja össze a látszólag áthidalhatatlan különbséget a képek digitális reprezentációja és az emberi értelmezés között. Ez a gát teszi kvázi lehetetlenné a bonyolultabb számítógépes látás problémák egyszerű algoritmussá történő megfogalmazását.

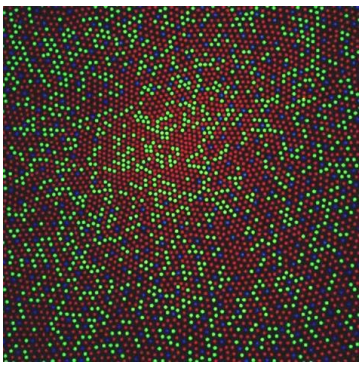
1.2. Képalkotás

A számítógépes látás legalapvetőbb feladata a feldolgozandó képek készítése. Bár a kamerák közismert, hétköznapi eszközök, ezek működésének és fontosabb tulajdonságainak ismerete elengedhetetlen ahhoz, hogy az adott alkalmazáshoz megfelelő eszközt válasszunk. Ebben a fejezetben a különböző szenzortípusok és az azok közti különbségeket mutatjuk be.

A kamerák működési elve nagy részben az emberi szem működési elvén alapszik. Az emberi szem egy üreges gömbtest, amelynek az elülső részén egy nyílás, a pupilla található, amelyen keresztül fény áramlik be a közvetlenül mögötte elhelyezkedő lencsébe. A lencse a párhuzamosan beérkező fénysugarakat egy pontba fókuszálja, amely pont a szem hátulján lévő fényérzékelő hártyán, a retinán helyezkedik el. A lencsének köszönhetően az egy irányból érkező fénysugarak a retinán pont ugyanarra a helyre érkeznek, így az emberi látás éles lesz.

A retina felületén számos fényérzékelő sejt helyezkedik el, amelyek továbbítják az ingereket az idegrendszernek. Fényérzékelő sejtből két féle létezik: a gyakoribb pálcikák csupán a fényintenzitást érzékelik, mivel a látható fény teljes tartományán hasonló érzékenységűek. Ezzel szemben a

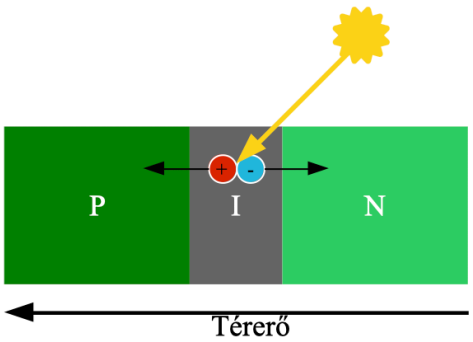
ritkábban elhelyezkedő csapok a kék, a zöld vagy a piros szín frekvenciatartományában érzékenyek csak, lehetővé téve a színek érzékelését.



1.4. ábra. *A csapok elhelyezkedése a retinán.*

1.2.1. Érzékelő típusok

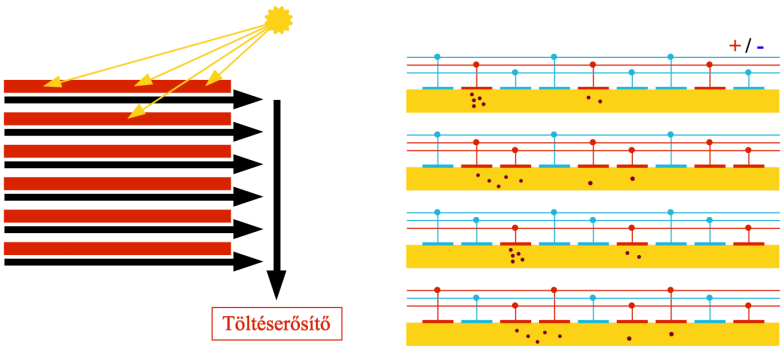
A kamerák az emberi látással analóg módon épülnek fel: geometriai kialakításuknak köszönhetően a beérkező fénysugarak egy szenzortömbre vetülnek, és az így kapott mért értékek fogják a képet alkotni. A kamerák geometriai felépítéséről egy későbbi előadásban beszélünk részletesen. A kamera fizikai felépítésén túl rendkívül fontos jellemző továbbá a szenzortömbben található fényérzékelő eszközök típusa. A fényérzékelés megvalósítását alapvetően fotodiódák segítségével végezzük. A fotodiódák a fotoelektromosság elvén működnek, vagyis, ha a dióda P-N átmenetére megfelelő energiájú fotonok csapódnak be, akkor ez egy elektron-lyuk párt fog gerjeszteni. Ha ez a dióda kiürített tartományára esik, akkor ezek onnan az elektromos tér hatására elmozdulnak, amely áramot eredményez.



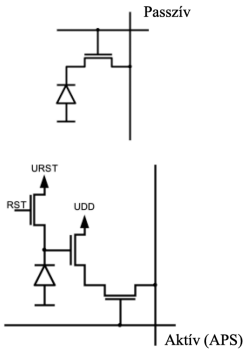
1.5. ábra. *A fotodióda működési elve.*

Fényszenzorok terén a két elterjedt megoldás a CCD-érzékelő (charge-coupled device), valamint az aktív CMOS-érzékelő (complementary metal-oxide semiconductor). A CCD-szenzor egyes elemei analóg eszközök, amelyek fotonok becsapódása esetén elektromos töltést tárolnak el. A kép készítéséhez szükséges idő eltelte után az egyes sorok legszélső cellája átadja a töltését a kimeneti erősítőnek, majd a cellák átadják töltésüket a szomszédos cellának. A fent leírt működési elv alapján a CCD-szenzor kiolvasása soronként, a sorokon belül pedig elemenként történik.

A CMOS-érzékelők esetén ezzel szemben minden cellára jut egy erősítő, így a szenzortömb kiolvasása lényegesen gyorsabb. Ennek viszont az az ára, hogy az erősítők helyet foglalnak a szenzortömbben, ami egyébként a fény elnyelésére szolgált volna. E hátrány kiküszöbölésére a CMOS-cellákra mikrolencséket tesznek, amelyek az érzékelőre terelik azokat a fotonokat, amelyek egyébként az erősítőre csapódtak volna be. Kisebb mérete, fogyasztása, valamint kedvezőbb ára miatt a kamerák többsége CMOS-érzékelőket használ. A CCD-szenzorok leginkább csúcsminőségű videokamerák terén elterjedtek.

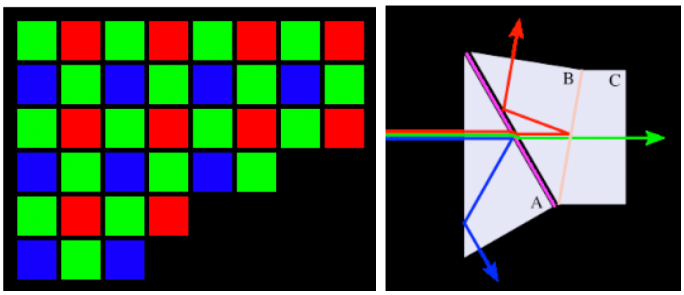


1.6. ábra. *A CCD szenzor működési elve.*



1.7. ábra. *A passzív és aktív cmos érzékelők felépítése.*

A kamerák színérzékelésének megoldására is több megoldás létezik, amelyek közül a legolcsóbb és leginkább elterjedt a Bayer-szűrőn alapuló megoldás. A Bayer-szűrő egy olyan tömb, amelynek egyes elemei csupán bizonyos színeket eresztenek át. Ezt a szűrőt a szenzortömb elé helyezve elérjük, hogy az egyes CCD- vagy CMOS-érzékelők is csak ezekre a színekre adjanak jelet. A Bayer-szűrő a legtöbb esetben kétszer annyi zöld színt áteresztő elemet tartalmaz, mint kéket és pirosat, aminek az az oka, hogy az emberi szem érzékenysége is hasonló.

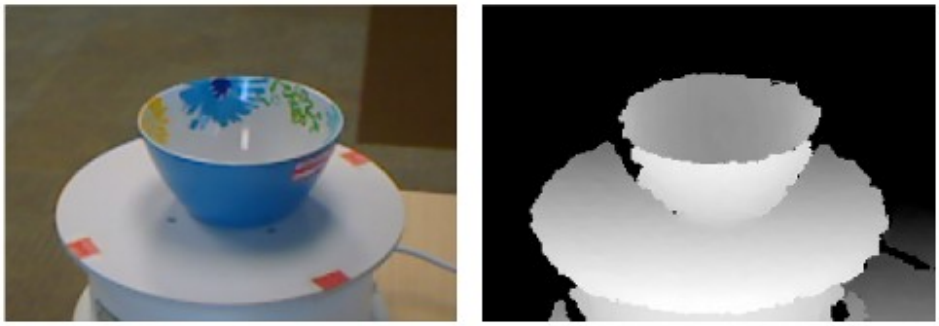


1.8. ábra. *A Bayer-szűrő felépítése (balra) és a 3CCD (jobbra).*

Egy alternatív megoldás a 3CCD-szenzor, ahol egy prizma segítségével a három színkomponenst külön-külön szenzortömbre tereljük, amellyel élesebb színelválasztást tudunk elérni. Mivel három külön szenzortömböt használunk, a 3CCD megoldás alacsony megvilágítás esetén lényegesen jobban teljesít a Bayer-szűrőnél, mindezt természetesen magasabb ár mellett.

Az elmúlt néhány évben egyre inkább elterjedtek olyan speciális szenzorok, amelyek az egyes pixelek intenzitása mellett azoknak a szenzortól számított távolságát is képesek meghatározni, így minden egyes képponthoz egy negyedik számértéket is hozzárendelnek. Ezeket az eszközöket Depth- vagy mélységkameráknak nevezzük.

E kameráknak alapvetően két változata létezik: az elsőt sztereokamerának nevezzük, ahol két, egymástól fix távolságra lévő kamera van egy házba építve, és az egyes pixelek távolságát a két



1.9. ábra. *Egy színes és egy mélység kép.*

készített kép közötti megfeleltetésekből számolhatjuk ki. Ezeknek az eszközöknek a kalibrációja általában a gyártás során megtörténik, valamint a mélység számítása a kamerába épített feldolgozó hardveren megtörténik.

Ezzel szemben az infravörös technológián alapuló mélységkamerák három elemből állnak: egy közönséges RBG-érzékelőből, egy infravörös vetítőből és egy az infravörös tartományban működő érzékelőből. A működésük elve az, hogy az ember számára láthatatlan infravörös tartományban egy előre meghatározott mintázatot vetítenek ki, amelyet az infravörös érzékelő visszaolvas, és a mintázat torzulásából következtet a látott kép térbeli struktúrájára. A kettő közül az infravörös alapú érzékelők elterjedtebbek, mivel jobb minőségű eredményeket adnak, és kevesebb feldolgozást igényelnek. Hátrányuk, hogy a környezetben található egyéb infravörös források megzavarhatják az eredményeket.

Létezik még ezen felül az úgynevezett LIDAR-érzékelő. Ez a radarral megegyező módon a visszavert elektromágneses hullámok késleltetéséből és frekvenciájából következtet az objektumok távolságára, csak éppenséggel rádióhullámok helyett lézersugarak segítségével működik. A digitális feldolgozóegységek válaszidejének hatalmas javulásának köszönhetően ma már rendkívül pontos távolságokat (néhány cm-es nagyságrend) tudunk e technológia segítségével mérni.

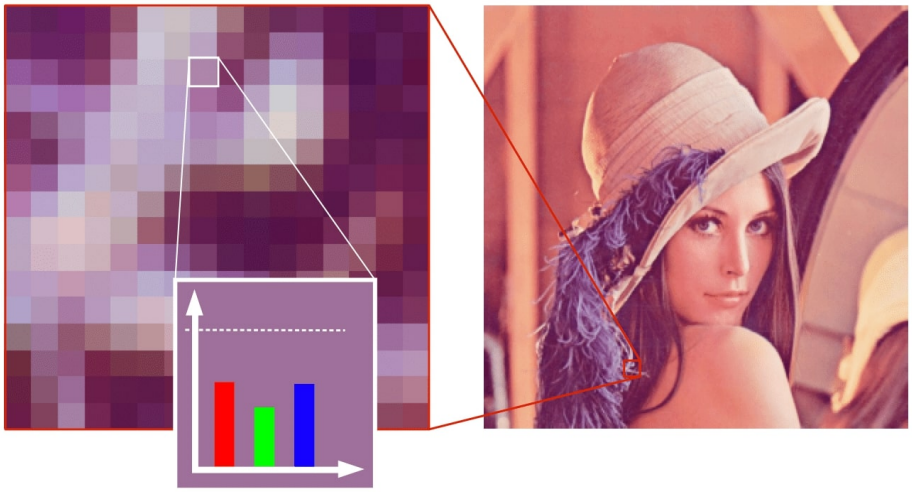
1.3. Képek tulajdonságai

Kiolvasás után a szenzorok által készített kép a számítógépbe beérkezve egy kétdimenziós számhalmaz lesz. Ennek egyes elemei az adott pozícióba beérkező fény intenzitását jelölik. Ezeket az elemeket képpontoknak vagy pixeleknek nevezzük. A számítógépben a pixeleket a legtöbb esetben egy 8 bites számmal jellemezzük, ahol 0 jelenti a teljesen sötétet, míg a maximális 255-ös érték pedig a teljesen világos képpontot. Színes képek esetén minden pozícióhoz három számérték tartozik, amelyeket RGB (red-green-blue) rövidítéssel jelölünk.

A digitális képeknek számos fontos paraméterük, tulajdonságuk van. Ezek közül az egyik legismertebb a felbontás, amely a számhalmaz dimenzióit adja meg. Ez a paraméter a kép részletességét nagy mértékben meghatározza, azonban növelésével a kép tárolásához szükséges adatmennyiség is növekszik. Fontos paraméter továbbá a képek bitmélysége, ami az egyetlen pixel tárolásához szükséges bitek számát adja meg. Leggyakoribb eset a 8-as bitmélység, lebegőpontos ábrázolás esetén azonban ez az érték 32. Különböző tömörítési eljárások során előfordul a 8-nál kisebb bitmélység is, ezek azonban a kép részletességéből valamelyest elvesznek. Egy bitmélységű képet bináris képnek nevezzük.

1.4. Tömörítés, tárolás

Képek tárolása esetén felmerülhet egy jelentős probléma: Egy átlagos mobiltelefon által készített kép ugyanis több tucat MB nagyságrendjében lévő információt tartalmaz, egy átlagos FullHD videó



1.10. ábra. *Egy kép felépítése.*

1.11. ábra. *A felbontás és a bitmélység csökkentésének hatása.*

pedig a több száz GB, vagy akár a TB nagyságrendjében is lehet. Ez azonban a mai merevlemezek kapacitását és sebességét is meghaladó terhelést jelentene. Éppen ezért fontos röviden szót ejteni a különböző tömörítő eljárásokról. Ezek általában az emberi látás sajátosságaira épülnek, lehetséges például egy pixelt 2 bájton tárolni úgy, hogy az egyes színkomponenesekhez rendelt bitszámot eltérő módon csökkentjük, kihasználva a színlátás sajátosságait.

Képek esetén számos elterjedt formátum létezik, melyek közül az egyik legalapvetőbb a BMP, amely egy tömörítetlen formátum. Léteznek veszteségmentes tömörített formátumok, melyek közül kiemelendő a png, amely egy raszter-grafikus formátum. Ezt a formátumot főleg szöveg és ábrák tömörítésére érdemes használni. Érdemes még megemlíteni az SVG és az EPS formátumokat, melyek alapvetően vektorgrafikus ábrázolást alkalmaznak.

Veszteséges tömörítések esetén a legelterjedtebb formátum a JPEG, illetve videók esetén az MPEG, melyeket tipikusan valós képekre/videókra alkottak meg. Videók esetén gyakori a H.264, illetve H.265 formátum, melyek az MPEG tömörítés legújabb változatai.

1.5. Színábrázolás

A számítógépes látás során gyakran használjuk ki a szürkeárnyalatos képekben rejlő intenzitás-információn felül a színes képek által hordozott extra információt is. Számos egyszerű detektáló

algoritmus épül színbeli hasonlóság alapú keresésre. Itt azonban számos problémába ütközhetünk. Például ahhoz, hogy a színbeli hasonlóság alapú keresés megbízhatóan, robusztusan működjön, arra van szükség, hogy a színeket leíró pixelértékek segítségével könnyen ki tudjuk fejezni a színek hasonlóságát.

A kamerák által leggyakrabban használt színábrázolás (vagy más néven az RGB-színtér) azonban erre nem alkalmas. Két színt leíró pont geometriai távolsága az RGB-színtérben nem kifejező arra nézve, hogy az emberi érzékelés mennyire érzi hasonlónak a két színt. Ezenfelül amennyiben a megvilágítási viszonyok megváltoznak, az egy RGB-kép esetén mindhárom értéket megváltoztatja, pedig a képen található objektum színe nem változott.

A színtértranszformációs eljárások célja, hogy az RGB helyett egy olyan új színreprezentációt adjanak meg, amely információ elveszítése nélkül képes a színbeli hasonlóságot jól leírni, és a megvilágítás változására is robusztus legyen. Ezek a transzformációk ezzel egyben egy új színteret is definiálnak. Maga a transzformáció bizonyos esetekben megadható egy mátrixszorzás segítségével.

$$\begin{pmatrix} S_1 \\ S_2 \\ S_3 \end{pmatrix} = \mathbf{C} \begin{pmatrix} R \\ G \\ B \\ 1 \end{pmatrix} \tag{1.1}$$

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \mathbf{C^{-1}} \begin{pmatrix} S_1 \\ S_2 \\ S_3 \\ 1 \end{pmatrix} \tag{1.2}$$

A leggyakoribb színtérkonverzió a szürkeárnyalatosítás, melynek során a három színcsatornából egyetlen fényesség jellegű értéket állítunk elő. Ennek számos módja létezik, melyek közül a leggyakrabban használtak az a luma(Y), az intenzitás (I), az érték (V) és a luminancia (L). Ezek előállítási módja a következő:

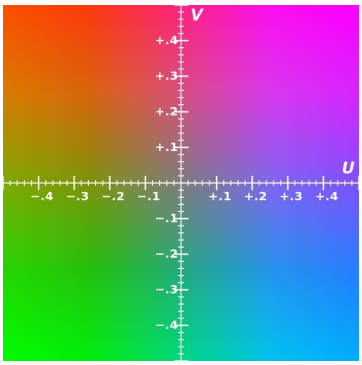
$$\begin{aligned} Y &= 0.3R + 0.59G + 0.11B \\ I &= \frac{1}{3}R + \frac{1}{3}G + \frac{1}{3}B \\ V &= max(R, G, B) \\ L &= \frac{max(R, G, B) + min(R, G, B)}{2} \end{aligned} \tag{1.3}$$

Az egyik leggyakrabban használt színtér az YCbCr-család, amelynek több, minimálisan különböző változata van. A színtér három csatornája közül az Y egy elkülönített világosságkomponens, amely az adott szín fényességét reprezentálja. A másik két csatorna, a Cr és a Cb pedig a szín árnyalatát kódolja el. Ezt a színteret gyakorta alkalmazzák digitálisvideó-rendszerekben, valamint a JPEG és az MPEG kódolása során. A gyakorlatban az YCbCr-színteret gyakorta összekeverik az YUV-színtérrel, amely hasonló elven működik, csak éppenséggel analóg rendszerekben használatos.

Az YCbCr áttérési mátrixa a következő:

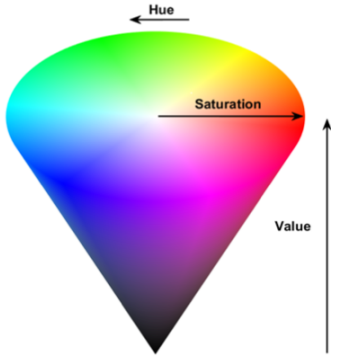
$$C_{YCbCr} = \begin{pmatrix} 0.299 & 0.587 & 0.114 & 0 \\ -0.169 & -0.331 & 0.5 & 128 \\ 0.5 & -0.419 & -0.081 & 128 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{1.4}$$

A másik, képfeldolgozás esetén gyakran használt család a HSV/HSI/HSL-család. E színterek közös jellemzője, hogy a színinformációt két érték, a hue (színárnylat) és a saturation (telítettség)



1.12. ábra. *Az LUV színtér UV síkja.*

segítségével írják le, míg a reprezentációk közötti alapvető különbség a fényesség/intenzitás reprezentációjának a módja. Ez a színtér könnyen ábrázolható egy henger vagy kúp formájában. A színalapú feldolgozás esetén rendkívül gyakori mindkét alternatív színtér használata.



1.13. ábra. *A HSV színtér egy kúp segítségével ábrázolható.*

Az HSV színtérre történő áttérés nemlineáris, vagyis nem írható le egyszerűen egy áttérési mátrixxal. A módszer a következő:

$$V = max(R, G, B)$$

$$S = \frac{V - min(R, G, B)}{V}$$

$$if \quad S == 0 \quad then \quad H = 0$$

$$c_r = \frac{V - R}{V - min(R, G, B)} \quad c_g = \frac{V - G}{V - min(R, G, B)} \quad c_b = \frac{V - B}{V - min(R, G, B)} \tag{1.5}$$

$$if \quad R == V \quad then \quad H = 0 + 60 * (c_b - c_g)$$

$$if \quad G == V \quad then \quad H = 120 + 60 * (c_r - c_b)$$

$$if \quad B == V \quad then \quad H = 240 + 60 * (c_g - c_r)$$

Az első ránézésre bonyolultnak tűnő szaturáció és Hue számítás mögött egyszerű elv rejtőzik: A szaturáció azzal arányos, hogy a legerősebb színkomponens mennyivel erősebb, mint a leggyengébb. Ezt követően a három színkomponens a kör három egyenlő távolságra lévő pontjára kerül: a piros a 0, a zöld a 120, a kék pedig a 240 fokos helyre. Ezt követően a két gyengébb színkomponens az aktuális színt a saját irányába igyekszik elhúzni.

További Olvasnivaló

[1] R. Szeliski, *Computer Vision*. Springer London, 2011. DOI: 10.1007/978-1-84882-935-0. cím: https://doi.org/10.1007%2F978-1-84882-935-0.

[2] L. Ross, "The Image Processing Handbook, Sixth Edition, John C. Russ. CRC Press, Boca Raton FL, 2011, 972 pages. ISBN 1-4398-4045-0(Hardcover)", *Microscopy and Microanalysis*, 17. évf., 5. sz., 843–843. old., 2011. szept. DOI: 10.1017/s1431927611012050. cím: https://doi.org/10.1017%2Fs1431927611012050.

2. fejezet

Képjavítás, szűrések

2.1. Intenzitás transzformációk

A képjavítás egyik legegyszerűbb módja a különféle intenzitástranszformációk használata. E módszer alkalmazása során az algoritmus pixelenként végighalad a képen, és minden egyes képpont értékét egy előre meghatározott transzformációs függvény alapján megváltoztatja. Ez a transzformációs függvény az esetek túlnyomó többségében az új intenzitásértéket csupán az adott képpont korábbi intenzitása alapján határozza meg.

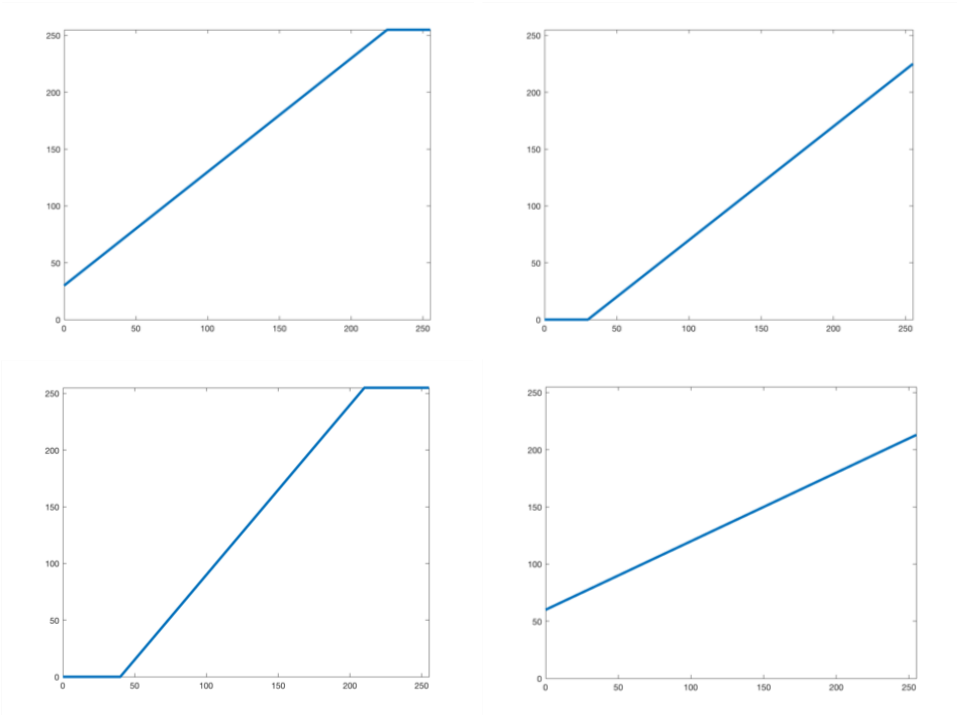
E transzformációknak több verziója létezik: amennyiben az egyes pixelek intenzitásához egy konstanst adunk hozzá, akkor a kép világosságát tudjuk változtatni (növelni vagy csökkenteni a konstans előjelétől függően). A pixelek világosságértékét egy konstanssal szorozni is lehetséges, ekkor a kép kontrasztosságát tudjuk csökkenteni vagy növelni. Kontrasztosságtranszformáció esetén egy konstans hozzáadása is megtörténik, hogy a kép új intenzitásértékeit középre toljuk. Fontos megjegyezni, hogy ezek a transzformációk feltételezik, hogy a pixelek értéke szaturál, azaz telítésbe megy, ha egy transzformáció azokat a minimális 0 érték alá vagy a maximális 255 érték felé vinné.

Az intenzitástranszformációk egy különös fajtája a küszöbözés. Ekkor egy előre meghatározott küszöbérték egyik oldalán lévő pixeleket 0-ba, míg a másik oldalon lévőket 1-be állítjuk, így egy bináris képet kapunk. Ezt a műveletet el lehet végezni két küszöbérték segítségével is, ekkor a tartományon belüli vagy kívüli értékeket állítjuk 1-re. A küszöbözés egyik fő haszna, hogy a képről bizonyos intenzitású képpontokat ki lehet emelni, és azok mennyiségéből, méretéből vagy pozíciójából további következtetéseket vonhatunk le. Fontos megjegyezni, hogy egy előre meghatározott küszöbérték használata esetén a fényviszonyok megváltozása a módszer eredményét is nagymértékben befolyásolhatja, így a gyakorlatban gyakran használunk adaptív – az adott képen lévő intenzitások eloszlásából meghatározott – küszöbértéket.

Intenzitástranszformációkat többcsatornás (vagyis színes) képek esetén is használhatunk. Ekkor ez egyes színcsatornákra egymástól függetlenül alkalmazunk transzformációs függvényeket. Ilyenkor természetesen más-más függvényeket használhatunk csatornánként, így lehetőség nyílik a képen a színek viszonyának, relatív dominanciájának változtatására. Küszöbözés használata esetén bizonyos színeket tudunk kiemelni a képről, így képesek lehetünk egy adott színű objektumot detektálni.

2.2. Hisztogram

Az intenzitástranszformációs módszerek rendkívül egyszerű és gyors módszerek, robusztusságuk azonban hagy némi kivetnivalót maga után. Éppen ezért gyakorta alkalmazunk képhisztogramra alapuló transzformációkat. A hisztogram egy adott képen vagy színcsatornán az egyes intenzitásértékek relatív gyakoriságát adja meg (ilyen értelemben az egyes intenzitások empirikus sűrűségfüggvénye). A hisztogram gyakran a segítségünkre lehet abban, hogy a képkészítés (általában a megvilágítás vagy az exponálás) hibáit detektálhassuk és javíthassuk.

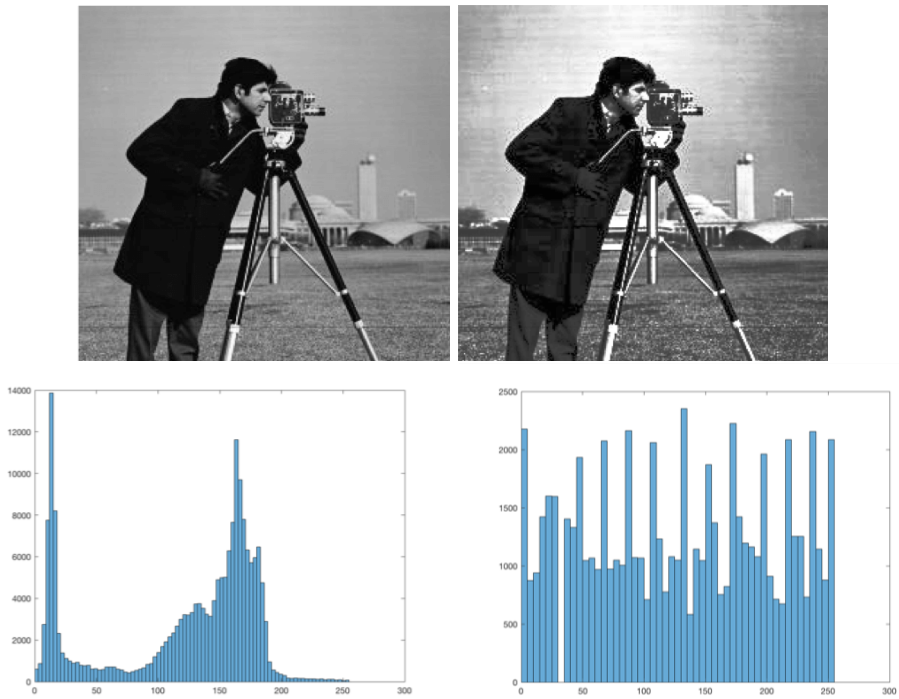


2.1. ábra. *Különböző intenzitástraszformációk: A világosság növelése és csökkentése (felül), illetve a kontraszt növelése és csökkentése (alul).*

2.2. ábra. *A küszöbözés eredménye.*

Alulexponált képek esetén a rövid expozíciós idő miatt nem csapódott be elég foton a szenzortömb egyes elemeibe, így a legvilágosabb képpont értéke is meglehetősen kicsi lesz. Ennek eredményeként a kép intenzitástartománya a teljes tartomány alsó felébe lesz összenyomva, ami miatt a kép részletei nehezen láthatók lesznek. Túlexponált képek esetén hasonló jelenség történik, csak a túl hosszú expozíciós idő miatt a sötét pixelek lesznek túl világosak, és a képi információ a tartomány felső részében lesz összenyomva.

Az ilyen jellegű hibákat kezeli a hisztogramkiegyenlítés algoritmusa. Alul- vagy túlexponált képek esetén a kép hisztogramja ugyanis jelentősen eltér az egyenletes eloszlástól. A hisztogramkiegyenlítés algoritmusának az a célja, hogy a ritka intenzitásértékek összevonásával és a gyakoriak mozgatásával az eredményként kapott hisztogram jobban közelítse az egyenletes eloszlást. Fontos megjegyezni, hogy az algoritmus alkalmazása során (ez egyes intenzitásértékek összevonása miatt) információt veszítünk, így a módszert csak a kép láthatóságának javítására szoktuk alkalmazni.



2.3. ábra. *Hisztogramkiegyenlítés: az eredeti kép és hisztogramja (balra), és a kiegyenlített kép (jobbra).*

2.3. Képi zajok, zajtípusok

A valós eszközökkel készített képek mindig zajjal és különböző hibákkal terheltek, amelyek a feldolgozást nehezítik. Ezek a zajok különböző forrásokból származnak, és ettől függően különböző típusai lehetnek. A képeken a leggyakoribb zajtípus a Gauss-zaj, amely a pixelszenzor saját belső zajának és az azt körülvevő elektronika zajának a következménye. Ez a zajtípus tipikusan additív, és pixelenként független.

A másik gyakori zajtípus még a só- és borszaj, amely az egyes pixelek értékében számottevő eltérést okoz, de csak ritkán fordul elő, így elszórt, sötét régiókban megjelenő világos pixeleket eredményez (vagy pont fordítva). Ezt a zajtípust leggyakrabban az analóg–digitális átalakító vagy az adásban bekövetkezett bithibák okozzák. Említésre méltó még a digitalizálás során keletkező kvantálási hiba, valamint az esetleges elektromágneses zavarások miatt fellépő periodikus hiba.

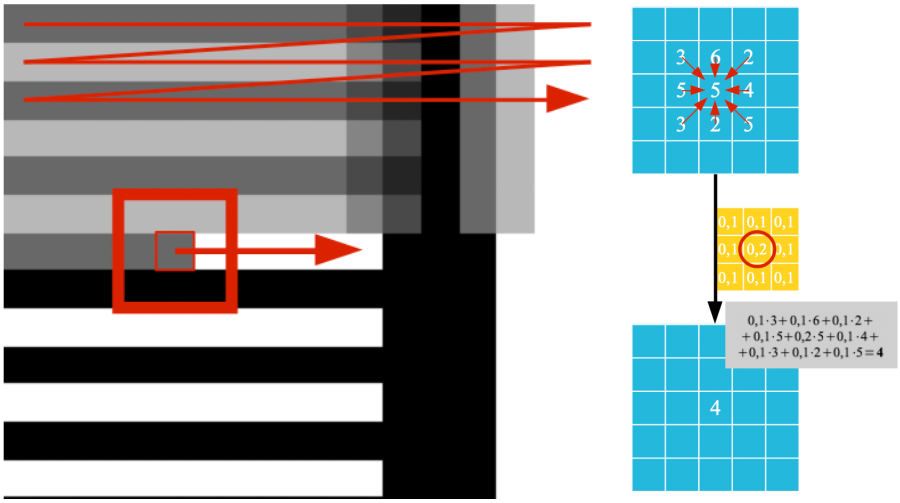
2.4. ábra. *A Gauss-zaj (balra) és a Só-bors zaj (jobbra).*

2.4. Konvolúciós szűrések

A képjavító algoritmusok családjának legfontosabb tagjai a különböző szűrő algoritmusok, amelyek a képi hibákat és zajokat hivatottak javítani. Ezek az eljárások konvolúciós szűrésen alapszanak. A képen egy kisméretű szűrőablakkal végighaladva, minden egyes pixelpozícióban az adott pixel új értéke a szűrőablak és a pixel lokális környezete között elvégzett konvolúció művelet eredménye lesz. A konvolúció műveletét az alábbi képlet adja meg:

$$(k \circledast I)(x,y) = \sum_{u=-n}^{n} \sum_{v=-n}^{n} k(u,v) * I(x-u, y-v) \tag{2.1}$$

Ahol $I(x,y)$ az y. képsor x. pixele. Mint a képletből is látható, a konvolúció művelete egyszerűen az adott környezetben lévő pixeleknek a szűrőből vett súlyok alapján számított súlyozott összege. A gyakorlatban mindig olyan szűrőket alkalmazunk, amelyeknél a súlyok összege egy, különben a képen világosabbá vagy sötétebbé tennénk. Fontos megjegyezni, hogy habár a konvolúció képlete alapján a képrészleten és a szűrőn ellentétes irányban kellene haladnunk, a gyakorlatban sokszor ezt mégsem így tesszük. Így a valóságban a keresztkorreláció műveletét számoljuk, de ezt mégis konvolúciónak nevezzük, holott a kettő eredménye csak középpontosan szimmetrikus szűrők esetén egyezik meg.

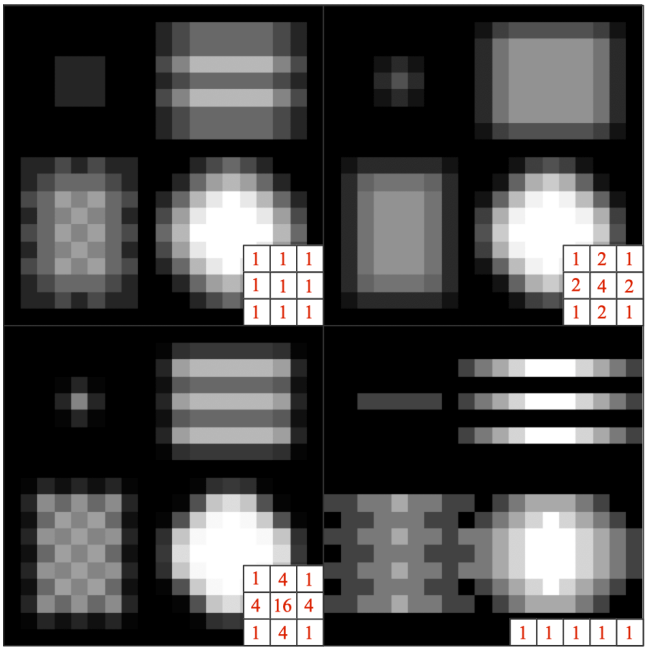


2.5. ábra. *A konvolúciós szűrés elve (balra) és a konvolúció művelete egy adott pozícióban (jobbra).*

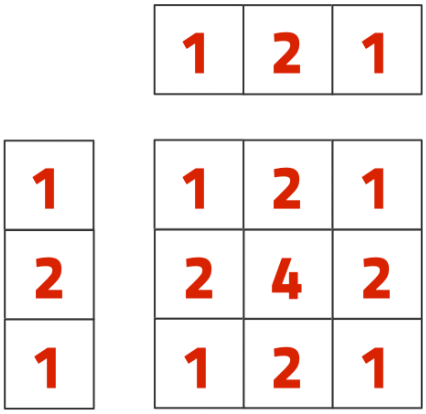
2.4.1. Lineáris szűrők

A zajszűrésre alkalmazott konvolúciós szűrőket simító szűrőknek nevezzük, amelyeknek legegyszerűbb változata az átlagoló szűrő. Valamivel kifinomultabb változat a Gauss-szűrő, amely a középponttól távolibb pixeleket kisebb súllyal veszi figyelembe, mindezt egy Gauss-haranggörbe alapján. A harangfelület szórásának állításával lehetőség van a simítás erősségének kézben tartására, sőt, ha az egyes irányokban más szórásértékeket adunk meg, akkor létrehozhatunk olyan Gauss-szűrőt, amely az egyik irányban sokkal drasztikusabban simít, mint a másikban.

A simító szűrők egyik legalapvetőbb tulajdonsága, hogy a konvolúciós ablak minden eleme nemnegatív, az eleminek összege pedig pontosan 1. Amennyiben a súlyok összege ettől eltér, a szűrő a simításon felül még világosítja, vagy éppenséggel sötétíti a képet. A konvolúciós szűrők egyik jó tulajdonsága, hogy lineáris műveletek, így egymás utáni szűrések könnyedén összevonhatók. Ezen túl bizonyos szűrőablakok esetén lehetőség nyílik a szűrő szeparálására: ekkor a 2D szűrést, két egymás után 1D szűréssel helyettesíthetjük. Ebben az esetben a szűrő végrehajtásának számítása N^2 helyett $2*N$ műveletbe kerül. Ezt azonban csak egy rangú szűrőmátrixok esetén tehetjük meg.



2.6. ábra. *Néhány tipikus konvolúciós szűrő: Az átlagoló (bal felül), a Gauss szűrő két különböző szórás értékkel (jobb felül és bal alul), valamint egy vízszintes átlagolást végző szűrő (jobb alul).*



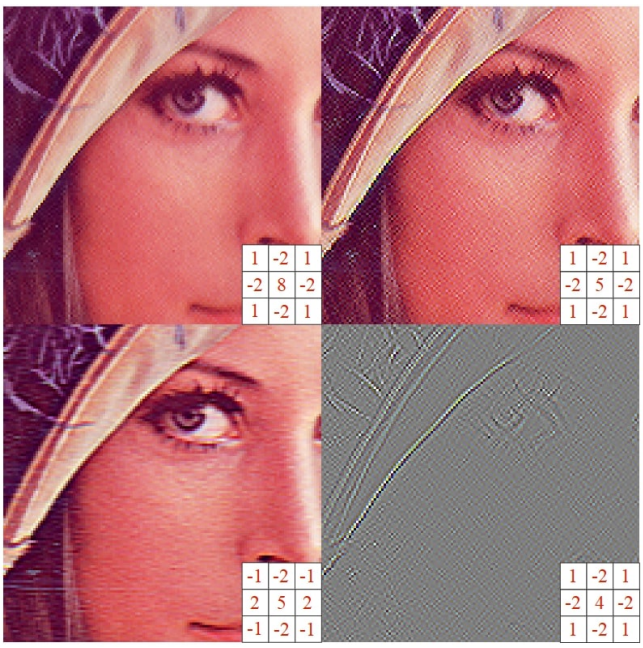
2.7. ábra. *Egy 2D szűrés fölbontva két 1D szűrőre.*

A konvolúciós simító szűrőknek két problémája van: az egyik, hogy az átlagolás után a zajokat ugyan hatékonyan eltüntetik, azonban a szűrés közben a kép egyes részleteit (főleg az éles váltásokat, éleket) is elmossák, ezzel homályossá téve a képet. Másrészt, mivel az összes ilyen szűrő valamilyen átlagolást végez, ezért az esetleges kiugró értékek (só- és borszaj) ezt az átlagot meglehetősen el fogják téríteni. Ennek eredményeképpen a só és bors jellegű zajokat ezek a szűrők inkább csak elkenik, ahelyett hogy kiküszöbölnék.

2.4.2. Élesítő szűrők

Megfigyelhető, hogy szemben a simító szűrőkkel, ahol a szűrő súlyok mindig pozitívak, léteznek olyan szűrők, ahol a szűrőben pozitív és negatív súlyok is vannak, mégis az értékeik összege 1. Ezeket élesítő szűrőknek nevezzük, és – mint azt nevük is sugallja – képesek a képeken a finom részleteket, változásokat kiemelni, ezzel a képet élesebb érzetűvé tenni. Ezt a szűrőfajtát gyakorta alkalmazzák fotós alkalmazásokban. Léteznek ezeken felül még ún. élkereső szűrők, melyek az élesítő szűrőkre hasonlítanak, azonban a súlyok összege 0.

2.8. ábra. *Egy fehér zajjal terhelt kép (balra) és a simító szűrés hatása (jobbra).*

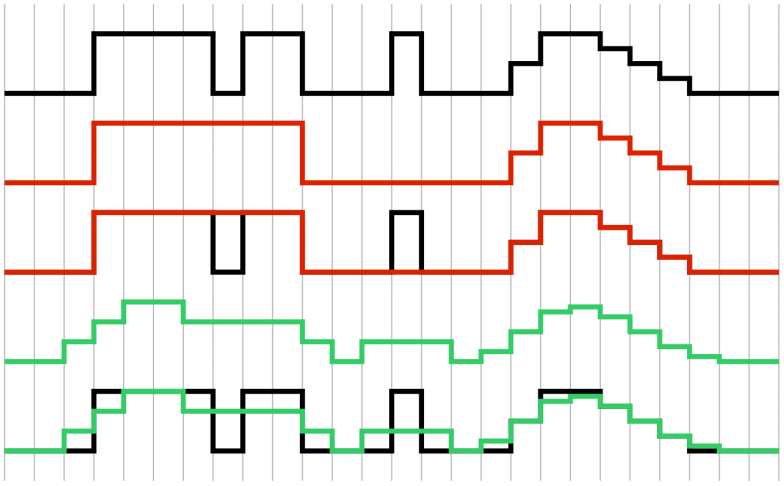


2.9. ábra. *Különböző élesítő szűrők, valamint egy élkereső szűrő (jobb alul). Figyeljük meg az eltérő szűrő összegeket.*

2.5. Rang szűrők

A simító szűrőkkel kapcsolatos, korábban felvetett problémákra adnak megoldást a rangszűrők. A rangszűrők szintén az adott pixel egy kis környezetét veszik figyelembe, azonban nem a konvolúció műveletét végzik el, hanem ehelyett a környezetben lévő pixeleket intenzitás szerint sorba rendezik, és a sorból egy értéket kiválasztva adnak új értéket az éppen vizsgált képpontnak. A rangszűrők közül a különböző feladatokra maximum, illetve minimum szűrőket szoktak használni, képszűrés esetén a mediánszűrők a legelterjedtebbek.

A mediánszűrők az adott pixelt a környezetükben lévő összes pixel intenzitása közül annak mediánjával, vagyis sorba rendezés után a középső értékkel helyettesítik. E szűrés rendkívüli előnye az, hogy az éles határvonalakat, éleket érintetlenül hagyja, míg rendkívül jól szűri a só és bors típusú zajokat. Ennek oka, hogy a mediánstatisztika az átlaggal szemben rendkívül robusztus a ritka, kiugró értékekre. A rangszűrők hátránya, hogy a sorba rendezés művelete rendkívül drága a konvolúcióhoz képest, így ezek a szűrők lényegesen lassabb működést eredményeznek. A helyzetet tovább rontja, hogy néhány magas szintű gyorsítási technika (szeparálható szűrők, frekvenciatartománybeli feldolgozás) is csak konvolúciós szűrőkkel végezhető el.



2.10. ábra. *A medián (piros) és az átlagoló szűrő (zöld) közti különbség.*

2.11. ábra. *Só és bors zajjal terhelt kép (balra) és medián szűrt változata (jobbra).*

2.6. Képi matematika

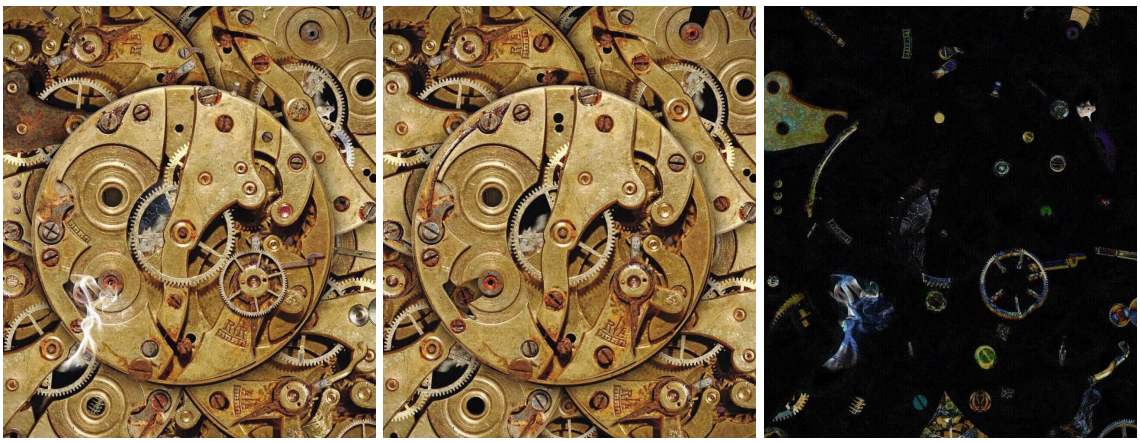
Mivel a képek kétdimenziós számtömbök, ezért reprezentációs szempontból a lineáris algebrából ismert mátrixokkal ekvivalensek. Ez azt jelenti, hogy bármi olyan művelet, amely mátrixokkal elvégezhető, képeken is működik. Ezek közül néhánynak (mátrixszorzás, felbontások) kevés értelme van, mások viszont kifejezetten hasznosak. Skalárműveletekkel megvalósíthatók a korábbi előadásban ismertetett intenzitástranszformációk: összeadással a világosság, míg szorzással a kontraszt állítható.

Azonos méretű képek között is végezhetünk műveleteket. Képek összeadása (vagyis inkább átlagolása) segítségével például blendelés végezhető, két kép elemenkénti szorzásának segítségével pedig texturázás valósítható meg. Az egyik leghasznosabb művelet azonban két kép kivonása. Ennek segítségével ugyanis a képek közti különbségek kiemelhetők, ami jelentősen megkönnyíthet számos feldolgozási feladatot. Ezt az eljárást gyakora használják mozgás detektálására, vagy a statikus háttér leválasztására.

2.7. Interpolációs technikák

A képek feldolgozása során gyakorta fordul elő olyan eset, amikor egy adott kép nem a megfelelő formában áll elő számunkra. Ezek közül gyakori eset, hogy a képet szeretnénk a felbontásából adódónál nagyobb méretben megtekinteni. A felbontás megnöveléséhez azonban meg kell határozunk, hogy az új pixelek értéke mi legyen. Erre természetesen extra információnk nincs, így kénytelenek vagyunk az ismert pixel értékeket és a képről alkotott feltételezéseinket (folytonosság,

2.12. ábra. *Textúrázás.*



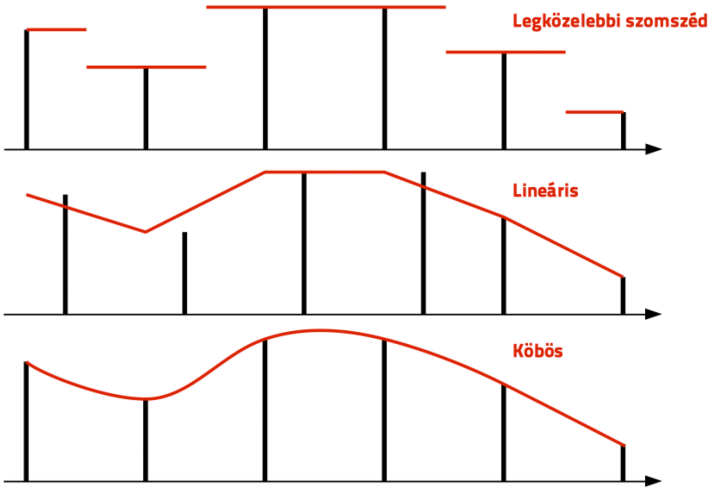
2.13. ábra. *Különbségképzés.*

simaság) felhasználni. Hasonló helyzetben vagyunk, ha a képeken geometriai transzformációkat (pl. forgatás, perspektív transzformáció) kívánunk végrehajtani, hiszen ebben az esetben az új, transzformált pixel rács nem fog az eredeti ráccsal tökéletes átfedésbe kerülni.

Az új pixel értékek meghatározásához úgynevezett interpolációs technikát szokás alkalmazni. Ezen technikák közös tulajdonsága, hogy egy jelet az ismert pontok felhasználásával egy előre meghatározott függvénnyel közelítjük, majd az új, még ismeretlen értékű pontok értékét ennek a függvénynek a mintavételezésével határozzuk meg. Érdemes megjegyezni, hogy ezáltal új információt nem vagyunk képesek teremteni.

Az interpolációnak számos különböző módszere van, amelyek különbsége a felhasznált függvénycsaládban rejlik. Ezek közül a legegyszerűbb a legközelebbi szomszéd (nearest neighbor) közelítés, amely a jelet szakaszonként konstans függvénnyel közelíti. Ekkor az új pixel értéke egyszerűen a legközelebbi szomszédja értékével fog megegyezni. Valamivel bonyolultabb a lineáris interpoláció, melynek során szakaszonként lineáris függvénnyel közelítünk, így az új pont értéke a két szomszédjának távolsággal súlyozott átlaga lesz. A közelítő függvény fokszáma természetesen tovább növelhető, elterjedt például a köbös interpoláció, amely az új pont négy legközelebbi szomszédjának felhasználásával becsült harmadfokú polinom segítségével végzi el az interpolációt.

A képek azonban kétdimenziós függvények, így az interpolációt is így kell elvégeznünk. Szerencsére a fent említett egyszerű interpolációs módszerek könnyű módon kiterjeszthetők tetszőleges



2.14. ábra. *1D interpolációs technikák.*

dimenziószámra. Ennek egyetlen korlátja, hogy az interpoláció elvégzéséhez szükséges szomszédok (és számítás) mennyisége a dimenziók számával exponenciálisan nő. Természetesen két dimenzióban is a legközelebbi szomszéd interpoláció a legegyszerűbb, ezt azonban csak mesterséges képek (geometriai ábrák, nyomtatott szöveg), vagy egyéb nem képi eredetű mátrixok esetében érdemes használni.

Az egyik leggyakrabban használt képi interpoláció a bilineáris interpoláció, amely a korábban említett lineáris módszer kiterjesztése. Ennek lényege, hogy az új pixel négy szomszédját felhasználva először két 1D interpolációt végzünk az egyik irányba. Ezt követően kapunk két új pontot, amelyek egyik koordinátája már megegyezik az új pontéval. Ezután nincs más dolgunk, mint e két pont között még egy - a másik irányba történő - interpoláció segítségével kiszámolni az új pont értékét. A lineáris interpolációnak két fontos előnye van: egyrészt egyszerű, másrészt nem okozhat túllövést. Hátránya azonban, hogy a keletkezett képet az emberi szem különösen homályosnak érzékeli, valamint az intenzitás meredekségében hirtelen változásokat okoz, amik a kép simaságát csökkentik. A bilineáris interpoláció eredménye az alábbi módon számolható:

$$\begin{aligned} f(x,y_1) &= \frac{x_2-x}{x_2-x_1}f(x_1,y_1)+\frac{x-x_1}{x_2-x_1}f(x_2,y_1) \\ f(x,y_2) &= \frac{x_2-x}{x_2-x_1}f(x_1,y_2)+\frac{x-x_1}{x_2-x_1}f(x_2,y_2) \\ f(x,y) &= \frac{y_2-y}{y_2-y_1}f(x,y_1)+\frac{y-y_1}{y_2-y_1}f(x,y_2) \end{aligned} \tag{2.2}$$

Természetesen a köbös interpoláció is kiterjeszthető két dimenziós esetre, amely 4 helyett már 16 szomszédos pixelt használ fel: ezt nevezzük biköbös interpolációnak. Ennek során a képet egy harmadfokú felülettel közelítjük, melynek képlete az alábbi:

$$f(x,y)=\sum_{i=0}^{3}\sum_{j=0}^{3}a_{ij}x^i y^j \tag{2.3}$$

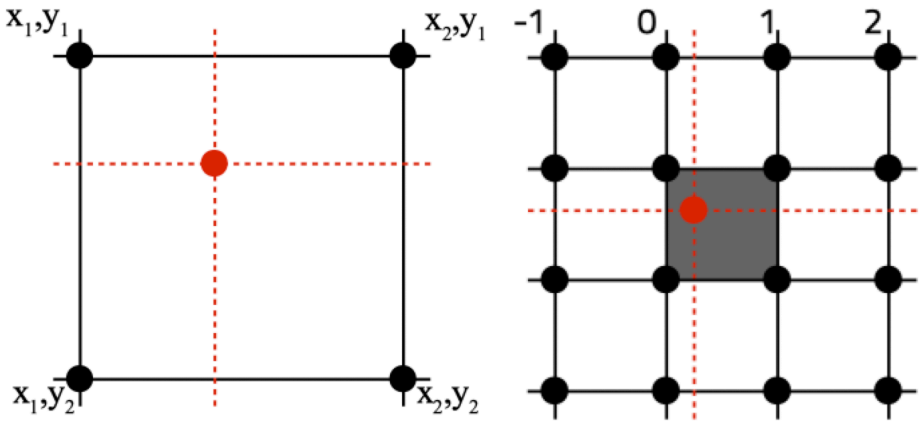
Ahol a_{ij} a polinom együtthatói. Az interpoláció feladata, hogy ezt a 16 ismeretlen együtthatót meghatározzuk. Ehhez azt kell tennünk, hogy az interpolálandó pixel körüli 2x2-es környezet mind a négy elemére felírunk 4 egyenletet. Egyrészt előírjuk minden pontban az intenzitás értékét, valamint előírjuk az intenzitás x és y irányú parciális deriváltjának értékét. Végezetül a közös parciális derivált (x és y szerinti) értékét is felírjuk, így a 16 paraméterre összesen 16 egyenletünk van, melynek segítségével ezek meghatározhatók. A felírandó egyenletek általánosságban az alábbiak:

$$\begin{aligned}\frac{\partial f(x,y)}{\partial x} &= \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij} i x^{i-1} y^{j} \\ \frac{\partial f(x,y)}{\partial y} &= \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij} x^{i} j y^{j-1} \\ \frac{\partial f(x,y)}{\partial x \partial y} &= \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij} i x^{i-1} j y^{j-1}\end{aligned} \tag{2.4}$$

Fontos megjegyezni, hogy alapesetben a deriváltak értékét nem ismerjük, ezek azonban a kép intenzitás értékeiből egyszerűen számolhatók az alábbi módon:

$$\begin{aligned}\frac{\partial f(x,y)}{\partial x} &= \frac{f(x+1,y) - f(x-1,y)}{2} \\ \frac{\partial f(x,y)}{\partial y} &= \frac{f(x,y+1) - f(x,y-1)}{2} \\ \frac{\partial f(x,y)}{\partial x \partial y} &= \frac{f(x+1,y+1) - f(x-1,y) - f(x,y-1) + f(x,y)}{4}\end{aligned} \tag{2.5}$$

A biköbös interpoláció egyik előnye, hogy alkalmazása után a kép élesebbnek tűnik, mint a bilineáris esetben, valamint, hogy a deriváltak széleken történő előírása miatt a kép sima marad. Hátránya azonban, hogy az interpoláció okozhat túllövéseket, amelyek az élek mellett gyűrű-szerű hibákat okozhatnak.



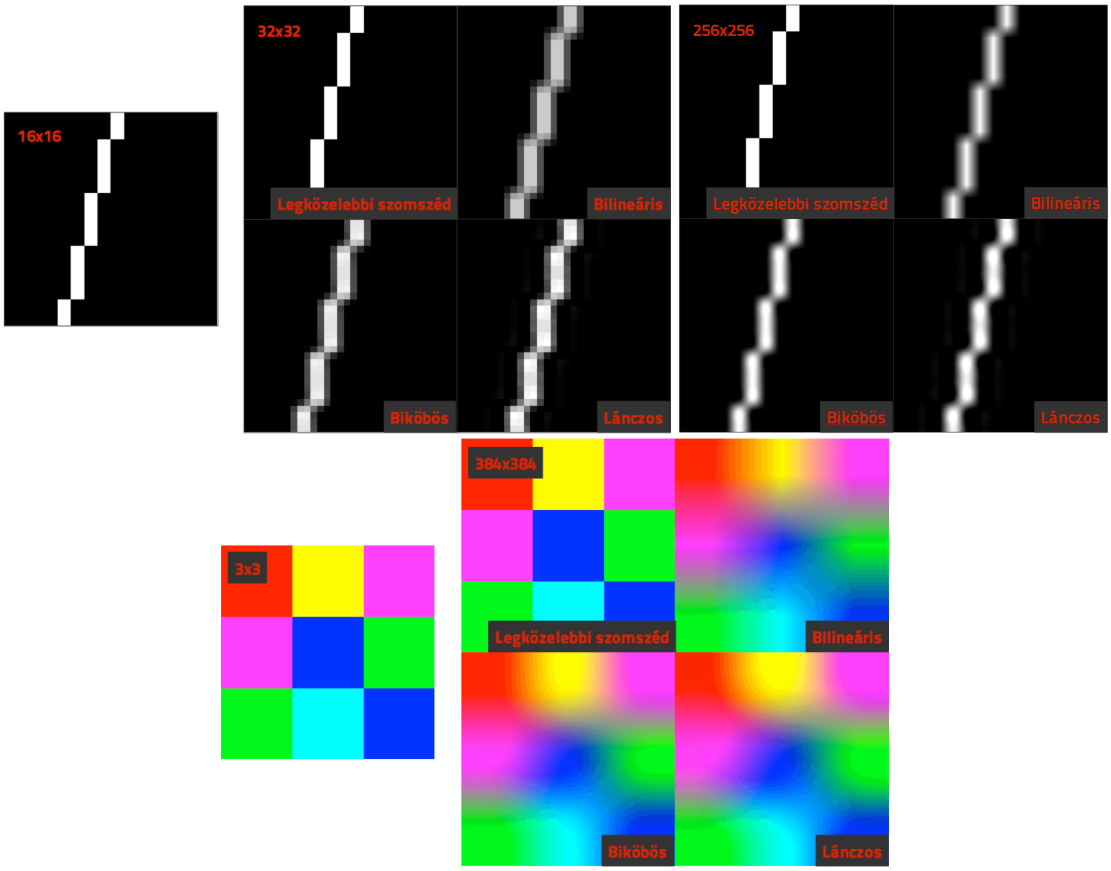
2.15. ábra. *A bilineáris (bal) és a biköbös (jobb) interpolációs technikák elve.*

Egy rendkívül jó kompromisszumot jelent a Lanzos interpoláció, amely a szomszédos pixelek ún. Lanzos-kernellel súlyozott átlagát veszi, így határozva meg az új értéket. Ez a módszer szintén jó élesség érzetet ad, azonban mindezt minimális túllövés mellett.

További Olvasnivaló

[1] R. Szeliski, *Computer Vision*. Springer London, 2011. DOI: 10.1007/978-1-84882-935-0. cím: https://doi.org/10.1007%2F978-1-84882-935-0.

[2] L. Ross, "The Image Processing Handbook, Sixth Edition, John C. Russ. CRC Press, Boca Raton FL, 2011, 972 pages. ISBN 1-4398-4045-0(Hardcover)", *Microscopy and Microanalysis*, 17. évf., 5. sz., 843–843. old., 2011. szept. DOI: 10.1017/s1431927611012050. cím: https://doi.org/10.1017%2Fs1431927611012050.



2.16. ábra. *A különböző interpolációs technikák eredménye különböző skálafaktor mellett.*

[3] J. Canny, "A Computational Approach to Edge Detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8 évf., 6. sz., 679–698. old., 1986. nov. DOI: 10.1109/tpami.1986.4767851. cím: https://doi.org/10.1109%2Ftpami.1986.4767851.

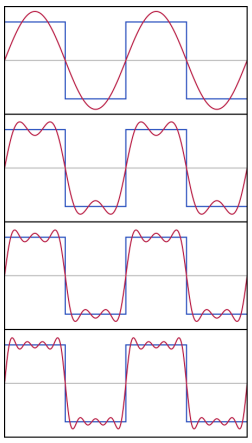
3. fejezet

Frekvenciatartománybeli feldolgozás

3.1. Bevezetés

A függvények tárgyalása során szinte magától értetődő, hogy a tér- és időfüggvényeket ezen mennyiségek segítségével ábrázoljuk közvetlenül. A függvényeknek azonban nem ez az egyetlen ábrázolási módja. A Fourier-sorfejtés tételének értelmében minden periodikus függvény felírható különböző frekvenciájú szinusz- és koszinuszfüggvények összegeként, ahol minden egyes frekvenciához tartozó függvényhez külön-külön amplitúdó (nagyság) és fázis (eltolás) tartozik. Ezeket a bizonyos frekvenciájú szinusz–koszinusz párokhoz meghatározott amplitúdókat és fázisokat (melyek egy komplex számként írhatók fel) nevezzük a jel spektrumának, a kép ezen értékek által történő megadását pedig a jel frekvenciatartománybeli ábrázolásának. A Fourier-sor az alábbi módon írható fel:

$$f(t) = a_0 + \sum_{k=1}^{N} a_k sin(k\omega_0 * t + \phi_k)$$
$$f(t) = \hat{f}_0 + \sum_{k=-N}^{N} \hat{f}_k e^{i*k\omega_0*t} \tag{3.1}$$



3.1. ábra. *A négyszögjel előállítása szinusz jelek segítségével.*

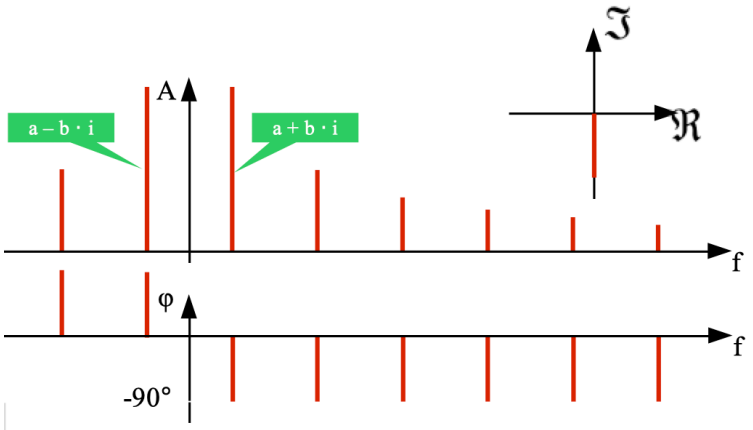
3.2. Fourier-transzformáció

A Fourier-sor legkisebb frekvenciájú tagja - az alap harmonikus - frekvenciája a jel periódusidejének reciprokával megegyezik, a sor többi tagja - a felharmonikusok - frekvenciája pedig ennek egész számú többszöröse. Könnyen belátható, hogy ha a jel periódusideje tart a végtelenbe (vagyis egy aperiodikus függvény felé), akkor a spektrum pedig egy folytonos függvénnyé fog válni. Ezt a folytonos komplex függvényt nevezzük egy tetszőleges jel Fourier-transzformáltjának. A Fourier-transzformáció mintavételezett (diszkrét) jeleken is elvégezhető, ekkor viszont a Fourier-transzformált lesz periodikus függvény - vagyis egy bizonyos frekvencia felett már nem tartalmaz új információt. Ez könnyen belátható módon azért van, mert egy mintavételezett jel nem képes akármilyen gyorsan változni.

A számítógépes látás tudományterületén szokványos egy kétdimenziós képet a képsík két dimenziójának függvényeként értelmezni. Ebben a vízszintes x és a függőleges y koordináták által kifeszített térben a kép diszkrét pontokban elhelyezkedő, pontszerű impulzusok összességeként írható fel, ahol az egyes impulzusok nagyságát az egyes pixelértékek adják meg. Szerencsénkre a korábban bevezetett Fourier-transzformáció azonban kiterjeszthető tetszőleges (esetünkben 2) dimenziószámra az alábbi módon:

$$F(u,v) = \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} I(x,y) * e^{-i(ux\frac{2\pi}{W} + vy\frac{2\pi}{H})} \tag{3.2}$$

Ahol $I(x,y)$ a pixelérték az y-adik sor x-edik oszlopában, u és v a vízszintes és függőleges irányú frekvenciakomponens, H és W pedig a kép magassága és szélessége.



3.2. ábra. *A diszkrét Fourier-transzformáció esetében adódó amplitúdó és fázis spektrumok.*

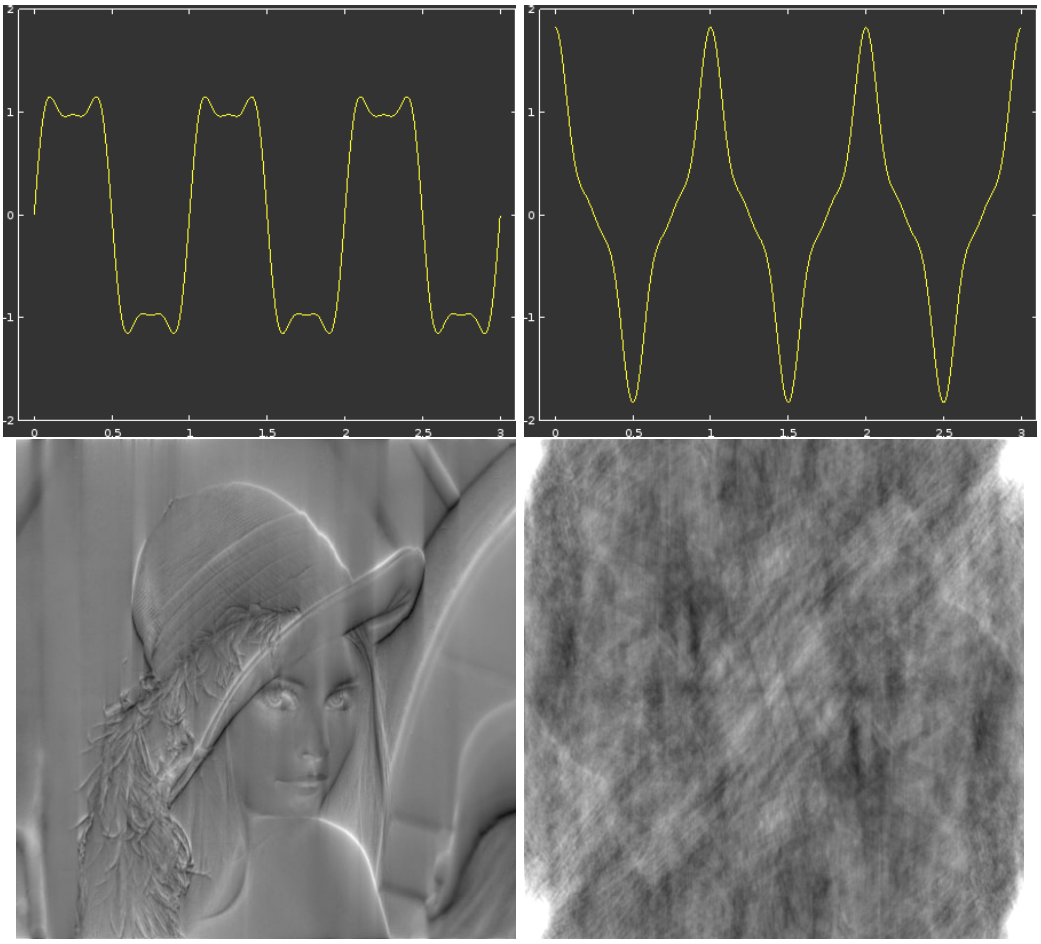
Mivel a képek kétdimenziós jelek, ezért a képet felépítő periodikus jeleknek iránya is van, nemcsak frekvenciája és fázisa (ezért is kétdimenziós a Fourier-transzformált). Így a megjelenített amplitúdóspektrum segítségével nemcsak a képen lévő domináns frekvenciákat, hanem azok irányát is megállapíthatjuk.

Korábban kifejtettük, hogy a Fourier-transzformáció fontos összefüggése, hogy a periodikus jelek spektruma diszkrét lesz, a diszkrét jelek spektruma pedig periodikus. Ez utóbbi számunkra szerencsés, mivel a kép is egy diszkrét jel, így a spektruma periodikus lesz, vagyis elég belőle egyetlen (véges méretű) periódust tárolni, így információ elvesztése nélkül tudjuk a képet frekvenciatartományba, majd visszakonvertálni. A számítógépek fizikai korlátai miatt azonban a kép spektrumát is csak diszkrét függvényként, mintavételezve tudjuk tárolni, így az összes frekvenciatartománybeli műveletünk azt fogja feltételezni, hogy a kép a szélein túl minden irányban periodikusan ismétlődik. Ez a viselkedés megváltoztatja, hogy egyes, egyébként ekvivalens algoritmusok miként működnek annak függvényében, hogy azokat kép- vagy frekvenciatartományban alkalmazzuk-e.

3.2.1. Fázistorzítás

Fontos megjegyezni, hogy a különböző jelek frekvenciatartománybeli tárgyalása esetében a fáziskarakterisztikáról lényegesen kevesebbet értekezünk, mint az amplitúdóspektrumról, mivel ez utóbbi fizikai értelmezése lényegesen szemléletesebb. Ez azonban nem jelenti azt, hogy a fáziskarakterisztika kevésbé lenne lényeges. A fázis különböző hibái, vagy torzítása ugyanis megegyező amplitúdóspektrum esetén jelentős eltérést tud okozni a jelalakban. Ennek oka, hogy a különböző felharmonikusok összeadódhatnak megegyező fázis esetében, amely hatalmas csúcsokhoz vezethet a jelalakban.

Ezt a jelenséget az alábbi képen figyelhetjük meg igazán jól: Amennyiben a négyszögjel felharmonikusait 90 fokos eltéréssel adjuk össze, akkor a négyszögjel megfelelő közelítését kapjuk, 0 fáziseltolás esetében viszont egy teljesen más jelet kapunk. Valós képek esetében ez könnyen a pixelértékek túlcsordulásával járhat, melynek következtében információt veszíthetünk.



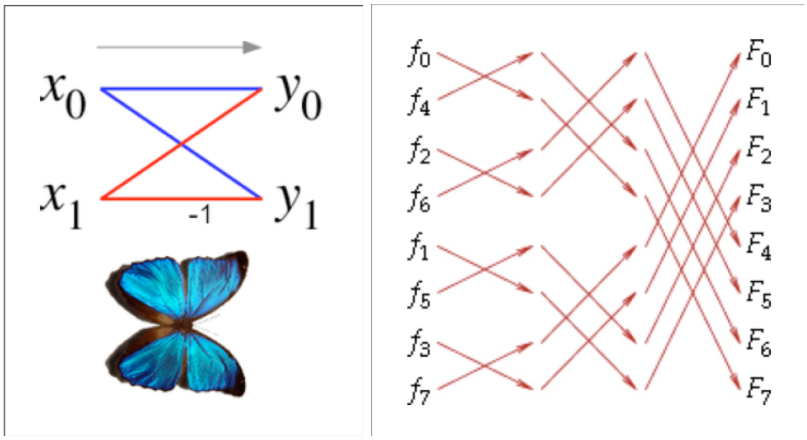
3.3. ábra. *A fázistorzítás hatása a négyszögjel esetében (felül) és egy valódi képen (alul).*

3.2.2. Gyors Fourier transzformáció

A kép spektrumát általában a gyors Fourier-transzformáció (FFT-Fast Fourier Transform) algoritmusával szoktuk meghatározni. Az FFT algoritmusának lényege, hogy a transzformálandó N elemből álló 1D függvényt rekurzív módon $\frac{N}{2}$ hosszú részekre osztja úgy, hogy a páros elemek az egyik, a páratlanok pedig a másik részbe kerüljenek. Ezt követően, ha már csak 2 elem van egy részben, akkor ezeken végrehajtja az úgynevezett pillangó műveletet.

$$y_0 = x_0 + x_1\omega^k$$
$$y_1 = x_0 - x_1\omega^k \tag{3.3}$$

Ezt követően a rekurzió folyamatában visszafelé haladva (egyre hosszabb, 2 hatványai szerinti) adatsorokon ismételten végrehajtjuk a pillangó műveletet, így végeredményként megkapjuk az N pontból álló FFT értékeket. Az algoritmus így a pillangók során kiszámolt részeredmények újrahasznosításával eléri, hogy a Fourier-transzformációt N^2 helyett $N\ \ log(N)$ lépésben kiszámolja. Érdemes megjegyezni, hogy a DFT-hez hasonló módon az FFT is könnyen kiterjeszthető 2 dimenzióra, amihez csupán a két irányban külön-külön kell egymás után elvégezni.



3.4. ábra. *A pillangó művelet (bal) és az FFT végrehajtása 8 adaton (jobb).*

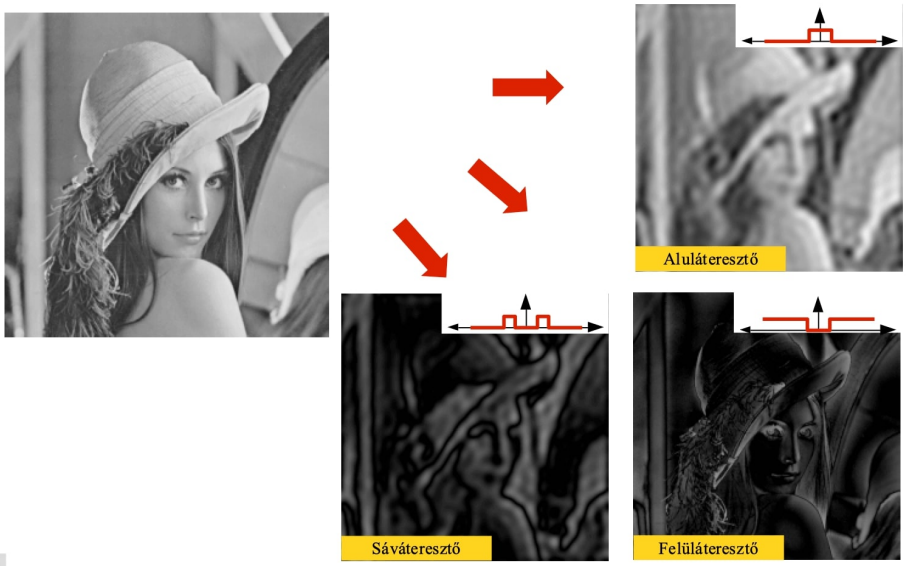
3.3. Szűrések

Az eddigiekben tárgyaltuk a frekvenciatartományban történő reprezentáció kérdését, azonban nem adtunk arra egy jó indokot, hogy mindezt miért érdemes megcsinálni. Ahogy azt az előadás hátralévő részében láthatjuk, a frekvenciatartománybeli képfeldolgozás egyik fő alkalmazási területe a különböző szűrések hatékony implementációja.

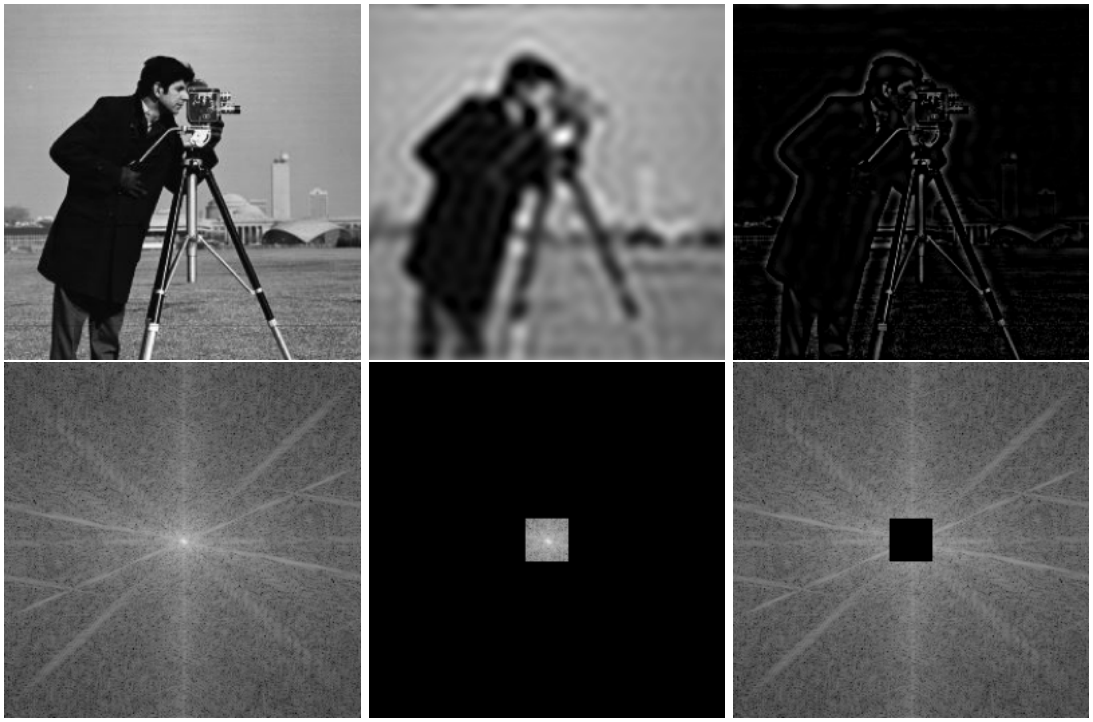
3.3.1. Ideális szűrők

Könnyű belátni, hogy ha egy képet fel lehet írni, mint különböző frekvenciájú szinusz- és koszinuszfüggvények összege, akkor e periodikus függvények közül az alacsonyabb frekvenciájú függvények a kép lassabban változó, simább jellemzőit foglalják magukban, míg a magasabb frekvenciájú komponensek a hirtelen, élesen változó részeket írják le. Ezt az összefüggést több célra is könnyedén kihasználhatjuk: a képi zajok pixelenként függetlenek, tehát hirtelen változásokat okoznak – tehát ha egy kép spektrumában egy aluláteresztő szűrő segítségével ezeket elnyomjuk, akkor zajszűrést végezhetünk. Hasonló módon, a képi élek hirtelen, éles változások az intenzitásfüggvényben, így ha a kép spektrumában egy felüláteresztő szűrővel csak a magasfrekvenciás komponenseket hagyjuk meg, akkor élkeresést végezhetünk. Ha a kép spektrumának a magasfrekvenciás komponenseit erősítjük, de az alacsonyfrekvenciás komponenseket nem töröljük el teljesen, akkor ez az élesítés műveletének felel meg.

Fontos kiemelni, hogy egy kétdimenziós kép Fourier-transzformáltja szintén egy kétdimenziós tömb lesz, amelyet lehetséges képként ábrázolni. Mivel minden frekvenciakomponenshez két érték tartozik, egy amplitúdó és egy fázis, ezért gyakorta szokás ezek közül csak az amplitúdót ábrázolni, mivel az az ember számára könnyebben értelmezhető információt tartalmaz. Az így megjelenített kép origójában található a nullafrekvenciás, azaz konstans komponens nagysága, ettől a kép szélei felé haladva pedig az egyre nagyobb frekvenciájú komponensek következnek. Fontos megjegyezni, hogy a kapott kép középpontosan szimmetrikus.



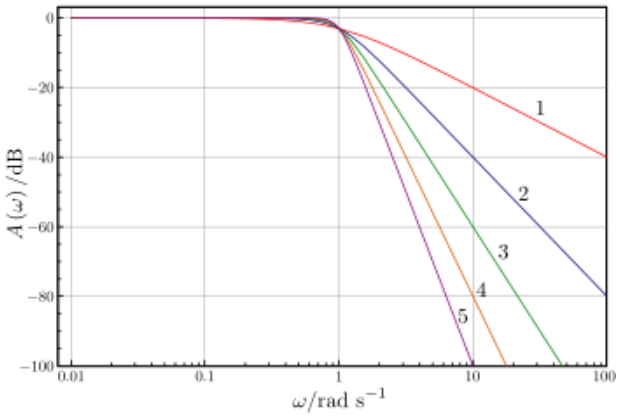
3.5. ábra. *Különböző szűrési módszerek a frekvencia-térben.*



3.6. ábra. *Szűrések és spektrumok. Eredeti kép és spektruma (bal), aluláteresztő szűrt kép és spektruma (közép), felüláteresző szűrt kép és spektruma (jobb).*

A fenti képeken könnyedén észrevehetjük, hogy az ideális aluláteresztő szűrő alkalmazásának van egy alapvető hátránya. Habár ez képek esetén az egyik legkönnyebben implementálható szűrőfajta (a szabályozástechnikával ellentétben itt a kauzalitás fogalmát idő dimenzió hiányában nem lehet értelmezni), a kapott képen különböző hullámzó gyűrű-szerű jelenségek keletkeznek. Ha visszaemlékszünk az előadás elején bemutatott négyszögjeles példára, akkor ennek oka nyilvánvaló: a nagyfrekvenciájú felharmonikusok teljes elhagyásával a néhány szinuszból ”összerakott” négyszögjelre hasonlító jelalakot kapunk.

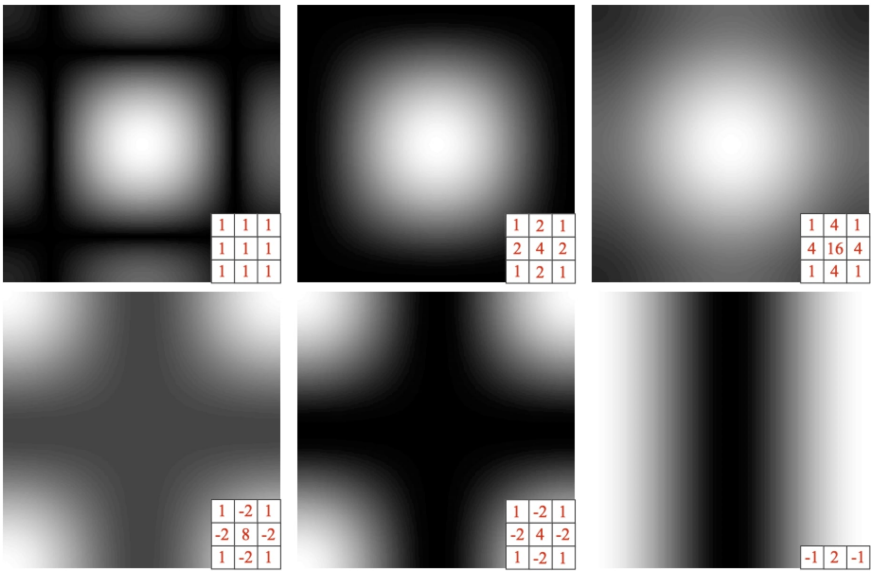
Ennek a problémának a megoldására általában valamilyen bonyolultabb spektrumú szűrőt szokás alkalmazni. Erre számos példát találhatunk, melyek közül a leggyakoribb az úgynevezett trapézszűrő, illetve annak ”sima” változata, a Butterworth szűrő.



3.7. ábra. *A Butterworth szűrő amplitúdóspektruma különböző fokszámok mellett.*

3.3.2. Konvolúciós szűrők

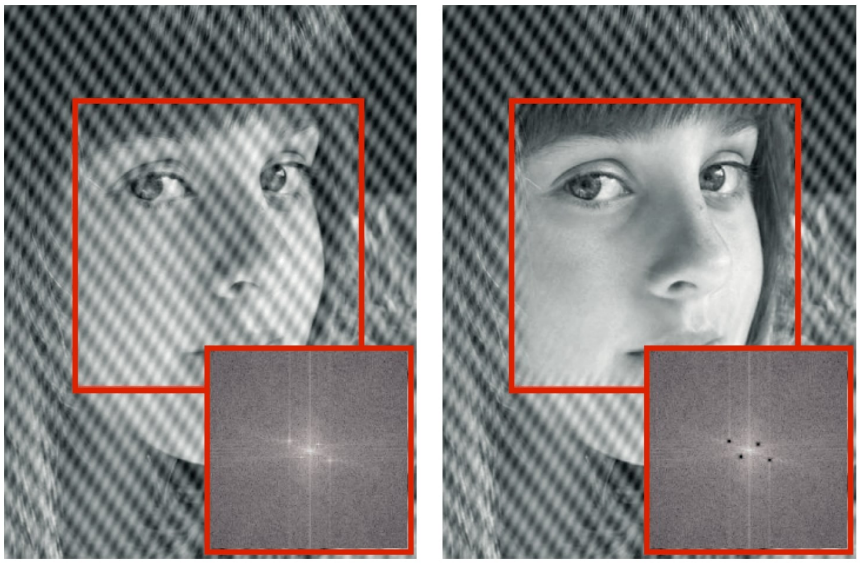
A Fourier-tér és a képtér közötti rendkívüli fontos összefüggés, hogy a képtérben elvégzett konvolúció művelete a frekvenciatartományban egy egyszerű elemenkénti (képek esetében pixelenkénti) szorzásra egyszerűsödik. Ez azt jelenti, hogy a különböző konvolúciós szűréseket lényegesen olcsóbb a frekvenciatartományban elvégezni. Ez különösen akkor előnyös, ha egy képen több szűrést is szeretnénk elvégezni, mert akkor a Fourier-transzformációt és annak inverzét csupán egyszer szükséges elvégezni, amelyek számítási költségét az olcsó szűrések megtérítik.



3.8. ábra. *Egyes konvolúciós szűrők spektruma.*

Korábban említést tettünk periodikus zajokról, amelyek általában valamilyen elektromágneses interferencia eredményeként keletkeznek a képen. Ezeket a zajokat simító szűrők segítségével nem lehet szűrni. Frekvenciatartományban azonban könnyedén megkereshetjük, és kiszűrhetjük a zajért felelős frekvenciát, és abból is elég csak a megfelelő irányút elnyomni.

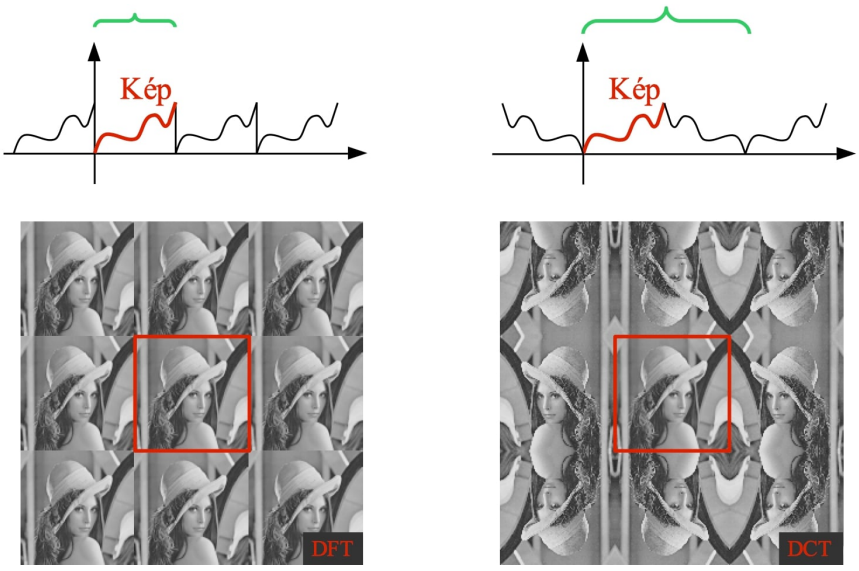
Alkalmazás: Egy másik felhasználási lehetőség az egyes dokumentumok (különösképp nyomtatott szövegek) irányultságának ellenőrzése. Egy nyomtatott szövegről készült kép esetén ugyanis a sötétebb sorok és világos sorközök egyenletes váltakozása egy jól kimutatható domináns frekvenciát fog létrehozni a kép spektrumában. E frekvenciakomponens irányát megfigyelve ellenőrizni tudjuk, hogy a szöveg vízszintesen látszik-e a képen, ami nagymértékben megkönnyíti például az optikai karakterfelismerő algoritmusok pontosságát.



3.9. ábra. *Periodikus zaj szűrése a frekvenciatérben.*

3.4. Koszinusz-transzformáció

Fontos megemlíteni egy, a Diszkrét Fourier-Transzformációhoz hasonló eljárást, amelyet Diszkrét Koszinusz-Transzformációnak (DCT) nevezünk. A DCT során a kép szintén periodikus jelként értelmezzük, azonban a kép szélein túl tükröződve ismétlődő jelalakot feltételezünk, és ezt a jelet transzformáljuk. Ennek következtében az eredeti jelünk szimmetrikus lesz, aminek következtében a kapott frekvenciaspektrum pedig csupa valós értékből fog állni.



3.10. ábra. *A DFT és DCT közti különbség.*

Ez a tulajdonság számos előnnyel jár számunkra, ugyanis a valós értékek eltárolása feleannyi memóriát igényel, így a kapott reprezentáció lényegesen tömörebb is lesz. Egy másik előny, hogy a tükröződő ismétlés miatt nem vezetünk be hamis ugrásokat a jelbe. DFT esetén ezek az ugrások megjelennének, mint nagyfrekvenciás komponensek, DCT esetében azonban ezek nem lesznek a spektrumban. További előnye ennek az eljárásnak, hogy a DCT meghatározása lényegesen egyszerűbb képlettel adódik:

$$D(u,v) = \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} I(x,y) * cos\Big[\frac{\pi}{W}(x+\frac{1}{2})u\Big] cos\Big[\frac{\pi}{H}(y+\frac{1}{2})v\Big] \tag{3.4}$$

3.4.1. FCT

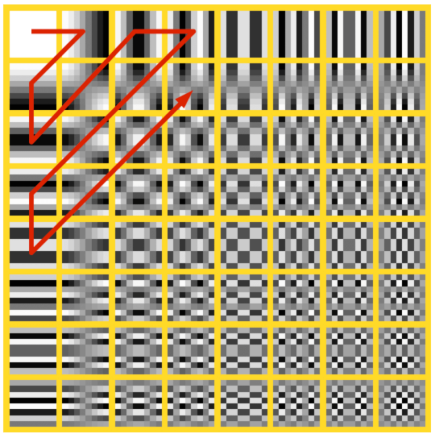
Érdemes megjegyezni, hogy az FFT művelete felhasználható a koszinusz-transzformáció kiszámítására, ehhez egész egyszerűen meg kell konstruálnunk a szimmetrikus képfüggvényt a tükrözések segítségével, majd az FFT algoritmusát ezen kell elvégezni. Ezt követően a kapott transzformáció valós részét véve megkaphatjuk a a koszinusz transzformált értéket. Érdemes megjegyezni, hogy léteznek olyan DCT-re specializált algoritmusok, amelyek nem végzik el a szimmetriából következő felesleges műveleteket.

3.4.2. JPEG

Érdemes megjegyezni, hogy az egyik legsikeresebb képtömörítési formátum, a JPEG is a DCT algoritmust, pontosabban annak II-es változatát használja fel a képek tömörítésére. A JPEG egy olyan veszteséges tömörítő eljárás, amely valós képek esetében gyakran képes 1:10-hez tömörítési arány elérésére úgy, hogy az információveszteség az emberi szem számára gyakorlatilag észrevehetetlen. Ehhez az emberi látás azon alapvető tulajdonságát használja ki, hogy a hirtelen intenzitásváltozásokat kevésbé vagyunk képesek érzékelni.

Az emberi látás hirtelen változásokra való érzékenysége ráadásul különbözik a szín és az intenzitás látás között. Mivel a színes fényt érzékelő csapok lényegesen ritkábban helyezkednek el a retinán, mint a pálcikák, ezért a színlátás felbontása még kisebb. Éppen ezért célszerű lenne a szín információt kisebb felbontáson tárolni, mint a világosság/fényerő komponenst. A hagyományos RGB képeken azonban ez a komponens nem különíthető el. Ennek megoldására a JPEG tömörítés YCbCr színteret használ úgy, hogy a Cb és a Cr színkomponensek felbontását az egyik, vagy leggyakrabban mindkét irányban megfelezi.

Ezt követően a JPEG algoritmus a képet 8x8 (felbontáscsökkentés esetében a színcsatornákon ez 16x16) méretű blokkokra osztja, majd a blokkokban lévő pixelekből azok átlagát levonja. Ezt követően minden blokkra külön elvégezzük a diszkrét koszinusz transzformációt, így elfőállítva azok frekvenciaképét.

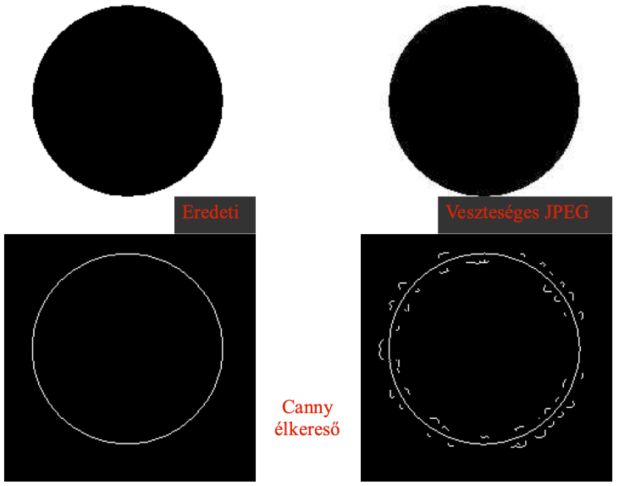


3.11. ábra. *A JPEG-tömörítés elve.*

Ezt követően a kapott lebegőpontos számokat egy a 8x8 méretű koszinusz transzformált minden eleméhez előre meghatározott konstanssal leosztjuk (az emberi szem sajátosságainak megfelelő konstansokról van szó, kihasználjuk ugyanis azt, hogy a frekvencia növelésével a kontrasztérzékelés romlik, így a nagyfrekvenciás együtthatók esetében jelentős helyet takaríthatunk meg), majd a

legközelebbi egészre kerekítve már elég minden elemhez csupán 8 bitet felhasználni. A kvantálás hatására azonban számos magasabb frekvenciájú elem nullázódni fog, így ezeket már nem kell külön eltárolni. Ennek maximális kihasználásához a transzformált elemein átlósan végighaladva indexelünk, így mindig a legnagyobb frekvenciákat hagyjuk utoljára. Ennek következtében elég egyszerűen a transzformált első néhány értékét eltárolni.

Érdemes megjegyezni, hogy a nagyfrekvenciás részek elhagyása következtében a JPEG a képi éleket hajlamos valamilyen szinten elmosni, valamint az ideális aluláteresztő szűrés mintájára az élek körül különböző gyűrűző képződmények (artifact-ok) jelennek meg. Ez ugyan szabad szemmel csak ritkán látható, egy élkereső algoritmus viszont könnyedén hamis éleket detektálhat ilyen képrészleteknél. Alapvetően elmondható, hogy mesterséges képek (geometriai ábrák, nyomtatott szöveg) esetében célszerűbb a png, eps, vagy hasonló tömörítések használata.



3.12. ábra. *A JPEG tömörítés hatása az élekre.*

3.5. Alkalmazások

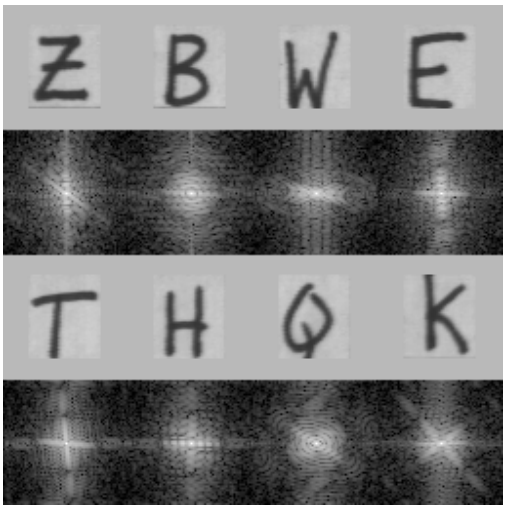
A szűréseken és tömörítéseken felül a frekvenciatartománybeli ábrázolásnak számos más alkalmazása is létezik. Ilyen például a különböző mozaik, vagy halftone képek valódi képpé történő konvertálása. Ilyen esetekben a mozaik elemeiből adódó ismétlődő mintázat általában rendkívül jól elkülönül a frekvenciatartományban, így itt könnyedén szűrhető.

3.5.1. Alakfelismerés

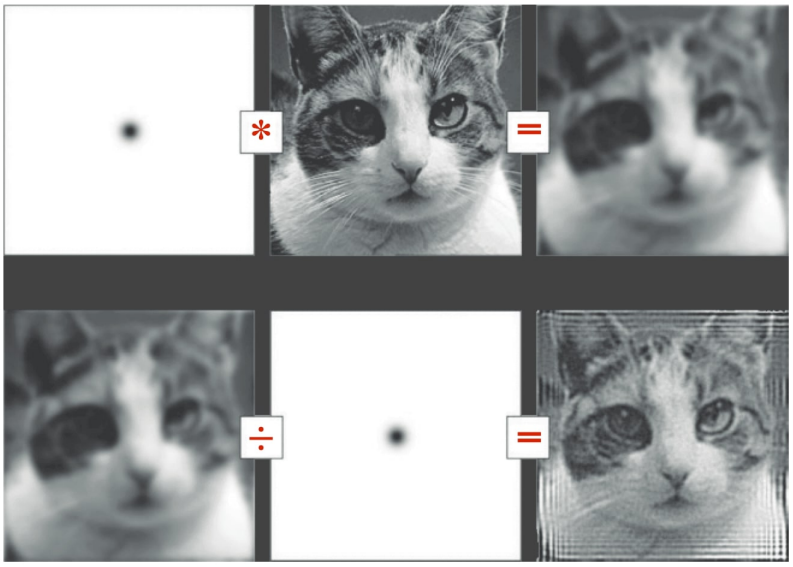
A Fourier transzformáció lehetőséget nyújt bizonyos egyszerű alakzatok felismerésére is. Erre jó példa az optikai karakterfelismerés (OCR) problémája. Az egyes nyomtatott karaktereknek ugyanis meglehetősen jól meghatározott alakja van, amely egyedi és szabályos spektrumot von maga után. Ez könnyedén felhasználható a karakterek felismerésére.

3.5.2. Dekonvolúció

Fontos megjegyezni, hogy a frekvenciatartománybeli ábrázolás lehetőséget ad a konvolúció műveletének egyszerű invertálására, melyet dekonvolúciónak hívunk. Dekonvolúció során a kép spektrumát nem szorozzuk, hanem osztjuk a szűrő spektrumával. Ezzel a módszerrel például kompenzálható a simító hatás, amit például a rosszul beállított fókusz okozott. A dekonvolúció egy fejlettebb változata a Wiener-dekonvolúció, amely a dekonvolúció melett egy frekvenciafüggő szűrést is végrehajt, ezzel elnyomva az esetleges fellépő zajokat.



3.13. ábra. *Különböző nyomtatott karakterek spektruma.*



3.14. ábra. *A Dekonvolúció elve és eredménye.*

További Olvasnivaló

[1] R. Szeliski, *Computer Vision*. Springer London, 2011. DOI: 10.1007/978-1-84882-935-0. cím: https://doi.org/10.1007%2F978-1-84882-935-0.

[2] L. Ross, "The Image Processing Handbook, Sixth Edition, John C. Russ. CRC Press, Boca Raton FL, 2011, 972 pages. ISBN 1-4398-4045-0(Hardcover)", *Microscopy and Microanalysis*, 17. évf., 5. sz., 843–843. old., 2011. szept. DOI: 10.1017/s1431927611012050. cím: https://doi.org/10.1017%2Fs1431927611012050.

[4] J. W. Cooley és J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series", *Mathematics of Computation*, 19. évf., 90. sz., 297–297. old., 1965. máj. DOI: 10.1090/s0025-5718-1965-0178586-1. cím: https://doi.org/10.1090%2Fs0025-5718-1965-0178586-1.

4. fejezet

Képjellemzők

4.1. Képillesztés, jellemző típusok

A képek számítógépes feldolgozása során az elsődleges feladatunk, hogy a képen olyan jellemző részleteket legyünk képesek megragadni, amelyek később felhasználhatók magasabb szintű feladatok végrehajtására. Ilyen képjellemzőkből számos fajta létezik, amelyek közül az egyiket – a képi éleket – az előző előadáson tárgyaltuk. A jelenlegi előadás témája a valamivel bonyolultabb, ebből következően számításigényesebb, azonban robusztusabb jellemzők tárgyalása.

4.1. ábra. *A képillesztés alkalmazásai: Objektumok azonosítása és relatív pozíciójának becslése (bal). Panorámaillesztés (jobb).*

4.2. Intenzitás

A különböző képi jellemzők legegyszerűbb fajtája maguk az intenzitás - vagy szín - értékek. Ezek meghatározásához nincs szükség külön algoritmusra, így előállításuk tulajdonképp költségmentes. Természetesen ezek a jellemzők egyben a legkevésbé robusztusak is, így csak speciális esetben érdemes használni őket.

4.2.1. Template matching

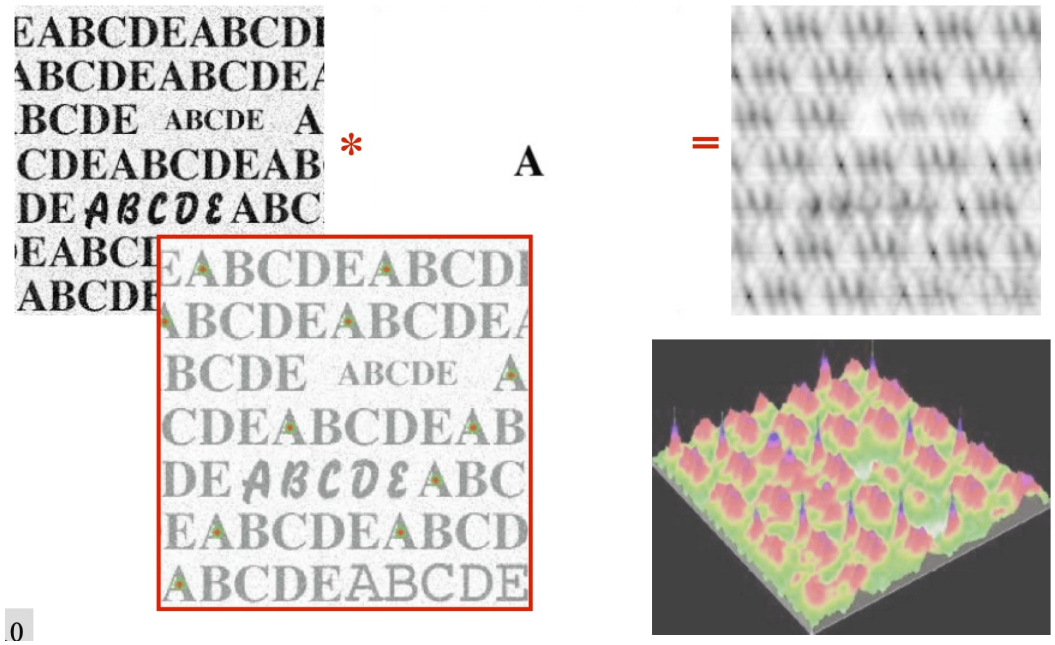
Ennek ellenére gyakran előfordulhat, hogy olyan feladatunk adódik, amikor relatíve kontrollált környezetben (statikus háttér és megvilágítás) egy előre ismert, nem változó objektumot kell detektálnunk. Ilyen esetekben használható a sablonillesztés (angolul: template matching) algoritmusa. Ez az eljárás rendkívül egyszerű: a detektálandó objektumról először referenciaképet készítünk, majd ezt a mintát a képre minden lehetséges pozícióban ráillesztjük, ezután pedig a kép és a sablon között valamilyen illeszkedési függvényt számolunk. Az illeszkedési függvény szélsőértékeinek pozíciójában pedig detektálást jelzünk.

A sablonillesztés során alapvetően kétféle illeszkedési függvényt használnak a gyakorlatban. Ezek közül az egyik a sablon és a képrészlet pixelei közötti négyzetes eltérések összege vagy más néven az L2 távolság, amelynek a minimumpontjait keressük. Érdekesebb megoldás azonban a konvolúciós/korrelációs illeszkedés, ahol a sablon és a kép közötti konvolúciót használjuk mérceként. A konvolúció alapvető tulajdonsága, hogy az eredménye akkor lesz abszolút értékben nagy, ha a szűrő és az általa lefedett képrészletre vagy annak inverzére hasonlít. Ennek illusztrálására említhetjük a korábbi előadáson bemutatott éldetektáló operátorokat, amelyek valóban úgy néznek ki, mint egy képi él.

$$
\begin{aligned}
E_{L2}(x,y) &= \sum_{x'}\sum_{y'}(I(x+x',y+y') - T(x',y'))^2 \\
E_{CC}(x,y) &= \sum_{x'}\sum_{y'} I(x+x',y+y')T(x',y')
\end{aligned}
\tag{4.1}
$$

A két hasonlósági mérce között lényeges különbség, hogy a konvolúciós megoldás esetén, ha a válasz mindkét szélsőértéke esetén jelzünk detektálást, akkor a sablon inverzét is detektáljuk. Ez hasznos lehet például betűk felismerésénél, ahol így egy fehér háttéren a fekete betűs mintát és a sötét háttéren a világos betűket is fel tudjuk ismerni. A sablonillesztési eljárás egyik főbb gyengesége, hogy a forgatásra, skálázásra és a torzításokra rendkívül érzékeny, így ha ezek előfordulnak, akkor minden skálához és orientációhoz külön sablont kell készíteni, és az eljárást az összes sablonnal meg kell ismételni, ami negatívan befolyásolja a sebességet.

Alkalmazás: A mintaillesztés eljárásának az egyik legfontosabb alkalmazása az optikai karakterfelismerés (OCR – Optical Character Recognition) azon esete, ahol nyomtatott karaktereket kell felismerni. Ebben az esetben a korábban ismertetett módszer segítségével, jó irányban beállított szövegre minden nyomtatott karakterhez egy külön mintát végigfuttatva az adott betű előfordulásait lokalizálni tudjuk, és így megkapható a szöveg. Az optikai karakterfelismerés alkalmazási területei pedig végtelenek, kezdve a szkennelt dokumentumok szöveggé konvertálásától a különböző igazolványok automatikus elolvasásáig.



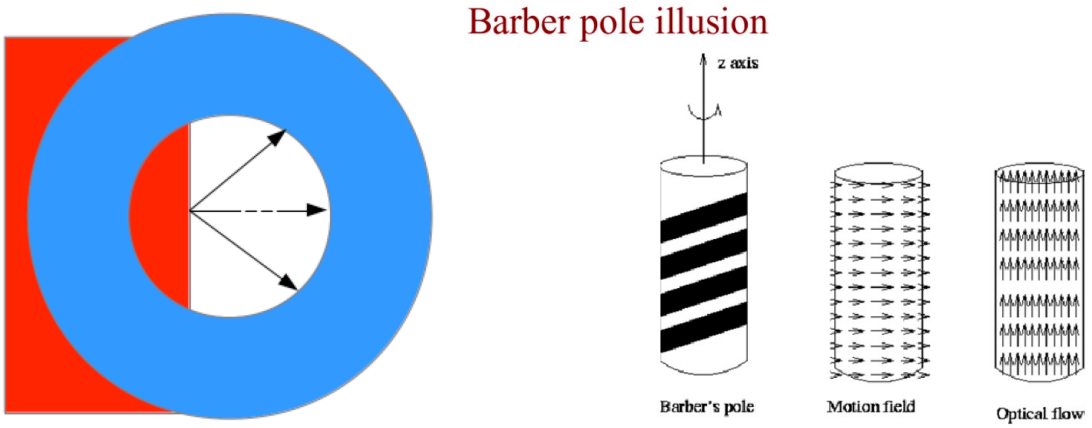
4.2. ábra. *A TM alkalmazása OCR területen.*

Fontos alkalmazás még a virtuális és kiterjesztett valóság tudományterülete, amelyben gyakorta valósítanak meg beviteli eszközöket oly módon, hogy az eszközökön valamilyen speciális (általában fekete-fehér) jelölő mintázatot helyeznek el. Ezeket a mintákat úgy szokták megválasztani, hogy azok a valóságos objektumokon ne fordulhassanak elő, így e markerek lokalizációja a mintaillesztés

segítségével egyszerűen és robusztusan megoldható. Ezt a módszert előszeretettel használják tapintható kiterjesztett valóság rendszerekben, amelyek alapvető elve, hogy a virtuális környezettel való interakciót speciális valós objektumok segítségével valósítják meg.

4.3. Optikai áramlás

Szintén az kép intenzitásértékeit használja jellemzőként a mozgás felismerése, szegmentálása és annak nagyságának és irányának meghatározására széles körben elterjedt optikaiáramlás-algoritmus. Az algoritmus alapelve, hogy a képet minden pixelben lokálisan egy lineáris síkfelülettel közelíti, majd a síkfelület meredekségéből és a két kép között történt intenzitásváltozás értékéből következtet a mozgás mértékére. Ennek egyik fontos következménye az úgynevezett apertúra probléma: bizonyos jellegű képrészletek esetén ugyanis nem lehet lokális képrészlet alapján a pontos elmozdulást meghatározni.



4.3. ábra. *Az apertúra probléma (bal). A lokális mozgásdetektálás az emberi látásban is szerepet kap: ezt jól illusztrálja a Barber-rúd illúzió (jobb). Itt egy ferde csíkokkal felfestett hengert körbe forgatva úgy látjuk, mintha a csíkok felfele haladnának, miközben a mozgás oldalirányú. Láthatjuk majd, hogy az optikai áramlás hasonlóan viselkedik.*

Az optikai áramlás feltételezi, hogy a képen az objektumok intenzitása nem változik a két kép között, csupán elmozdulás történik. Ezt felírhatjuk az alábbi módon, a képet Taylor-polinommal közelítve:

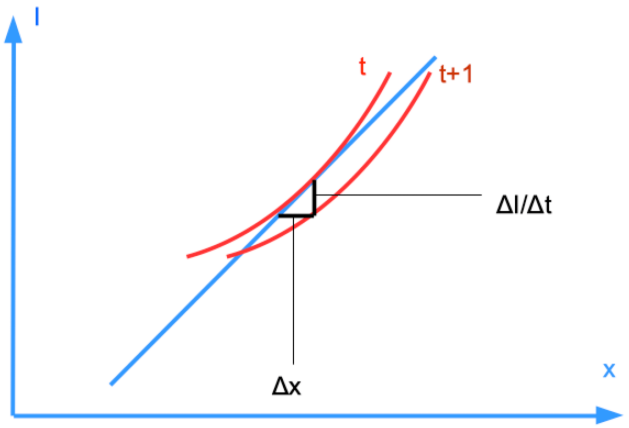
$$I(x,y,t) = I(x+dx, y+dy, t+dt) = I(x,y,t) + \frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt \tag{4.2}$$

Ahonnan $I(x,y,t)$ kiesik:

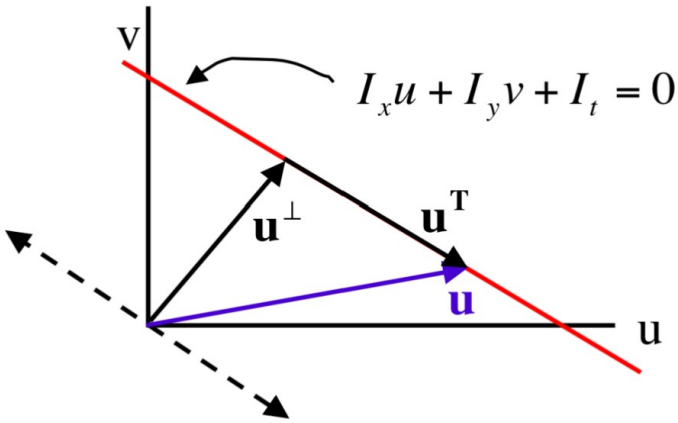
$$\begin{gathered} I_x dx + I_y dy + I_t dt = 0 \\ I_x \frac{dx}{dt} + I_y \frac{dy}{dt} + I_t = I_x u + I_y v + I_t = 0 \end{gathered} \tag{4.3}$$

Ahol I_x, I_y és I_t a kép irányok és az idő szerinti deriváltjai, u és v pedig a mozgás sebessége az egyes irányokban. A kép deriváltjait numerikusan szűrők és különbségképzés segítségével tudjuk számolni, azonban így is egyetlen egyenlet jut két ismeretlenre, amiből következik, hogy az intenzitásáramlás-egyenlet nem oldható meg egyértelműen. A valóságban az egyszerű áramlásmódszer csupán a mozgásvektornak az egyik – a képi gradiens irányába eső – komponensét képes meghatározni az alábbi módon:

$$d = \frac{I_t}{\sqrt{I_x^2 + I_y^2}} \tag{4.4}$$



4.4. ábra. *Az optikai áramlás meghatározása egy dimenzióban: lokálisan egyenessel közelítjük a képet, így az adott pixel időbeli változását ismerve a becsült elmozdulás egyszerűen számítható.*



4.5. ábra. *Az intenzitás-áramlás egyenlet által meghatározott megoldáshalmaz (piros egyenes), a megoldás komponensei.*

4.3.1. Lucas-Kanade módszer

A gyakorlatban erre a problémára az egyik széles körben használt megoldás a Lucas–Kanade-féle optikaiáramlás-algoritmus. Ez az eljárás eggyel több feltételezést eszközöl, mégpedig azt, hogy a képen egymáshoz közel található pixelek nagy valószínűséggel együtt is mozognak. Ennek alapján a Lucas–Kanade-módszer nemcsak az adott pixel pozíciójában kívánja megoldani az intenzitásáramlás-egyenletet, hanem annak egy bizonyos környezetében. Erre a legkisebb négyzetek módszerét alkalmazza az alábbi módon:

$$\begin{pmatrix} I_{x1} & I_{y1} \\ I_{x2} & I_{y2} \\ \vdots & \vdots \\ I_{xn} & I_{yn} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} I_{t1} \\ I_{t2} \\ \vdots \\ I_{tn} \end{pmatrix} \tag{4.5}$$
$$X\vec{u} = Y$$

Itt a fenti egyenletben a deriváltak ismertek, az elmozdulás vektort keressük úgy, hogy a fenti egyenlet a lehető legkisebb hiba mellett kerüljön megoldásra. Ehhez az egyenletrendszert a legkisebb négyzetek módszerével kell megoldanunk, mely esetben az optimális elmozdulás az alábbi:

$$\vec{u} = (X^T X)^{-1} X^T Y \tag{4.6}$$

Fontos megjegyezni, hogy az egyenletben szereplő $X^T X$ mátrix a korábban ismertetett KLT-sarokdetektorban is használt lokálisstruktúra-mátrix (ez nem meglepő, hiszen a KLT-detektor nevében szereplő Kanade és Lucas ugyanazok a kutatók, akikről a jelenleg tárgyalt módszer is az elnevezését kapta). Ha visszaemlékezünk a lokálisstruktúra-mátrixról folytatott diszkusszióra, tudhatjuk, hogy ennek a mátrixnak a sajátértékei a kép legkisebb és legnagyobb változásának nagyságát fejezik ki. Ahhoz, hogy ezt a mátrixot invertálni tudjuk, mindkét sajátértéknek jelentősen különbözni kell a nullától, vagyis a Lucas–Kanade-módszert csak sarokszerű pontok esetén lehet kiértékelni. Éppen ezért ez a módszer ritka optikaiáramlás-módszerként is ismert, mivel csak a kép néhány pontján számolható ki.

4.3.2. Farneback módszer

Létezik azonban sűrű optikaiáramlás-algoritmus is, amely képes egyértelműen meghatározni az áramlás irányát, és minden pontban számítható is. Ez az eljárás Gunnar Farneback nevéhez fűződik. Az eljárás alapvető lényege, hogy a képet lokálisan nem egy lineáris felülettel (síkkal), hanem egy másodfokú polinommal közelíti. Az eredeti áramlásalgoritmushoz hasonlóan feltételezi, hogy a két képkocka között nem történik intenzitásváltozás, csupán elmozdulás. Innen a polinomokat mindkét képkockán kiszámolva és összevetve az egyes pixelek elmozdulása meghatározható.

$$I_1(x) = x^T A_1 x + b_1^T x + c_1; \quad I_2(x) = x^T A_2 x + b_2^T x + c_2 \tag{4.7}$$

Felhasználva azt a feltételezést, hogy a két kép megegyezik, köztük csupán egy d eltolás van:

$$I_2(x) = I_1(x - d) = (x - d)^T A_1 (x - d) + b_1^T (x - d) + c_1 \tag{4.8}$$

Ezt x hatványai szerint csoportosítva:

$$I_2(x) = x^T A_1 x + (b_1 - 2A_1 d)^T x + d^T A_1 d - B_1^T d + c_1 \tag{4.9}$$

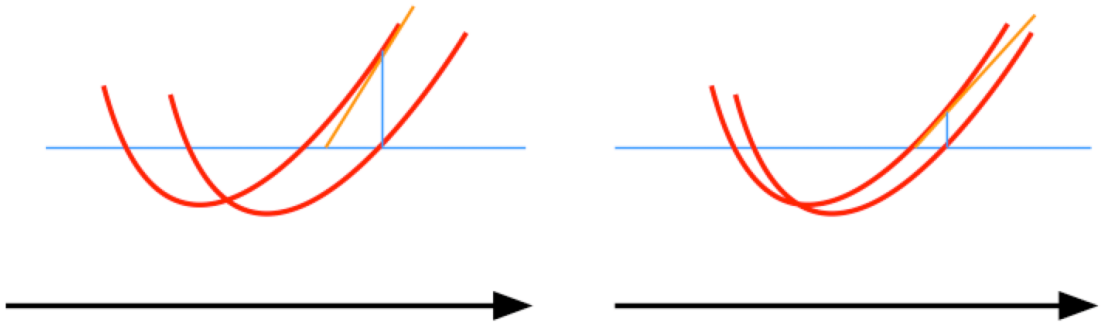
Ebből pedig felírhatjuk, hogy a két polinom b együtthatói megegyeznek, amiből a d kifejezhető:

$$b_2 = b_1 - 2A_1 d \quad \rightarrow \quad d = -\frac{1}{2} A_1^{-1} (b_2 - b_1) \tag{4.10}$$

A gyakorlatban azonban egy enyhén módosított változatot alkalmazunk: a kép ugyanis globálisan nem közelíthető jól egy másodfokú polinommal, így ezt a közelítést lokálisan alkalmazzuk, és az elmozdulást minden pozícióban határozzuk meg. Fontos még megjegyezni, hogy a Farneback alkalmazásakor az egyes pozíciókban elvégzett polinombecsléshez a környezet pixeleit is felhasználjuk. Ily módon a legkisebb négyzetek (LS) módszerével egy sokkal robusztusabb becslést kapunk, így kiküszöbölve a zajok hatását.

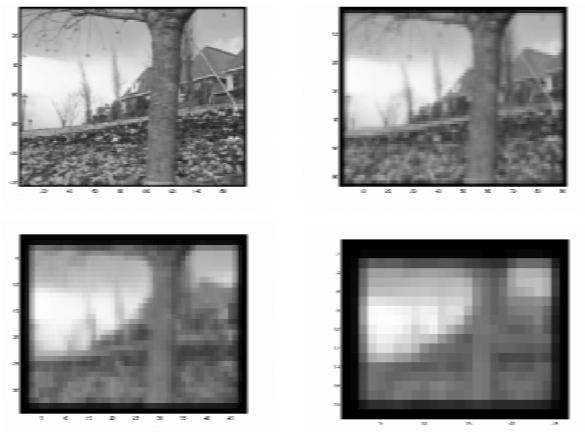
4.3.3. Iteratív és piramis módszerek

Az optikai áramlás módszerek esetében azonban jelentős problémát jelent az, ha az elmozdulás annyira nagy, hogy a magasabb rendű tagok már nem elhanyagolhatók. Ekkor ugyanis a mozgásbecslés hibája már számottevő lesz, ami hosszabb távon teljesen rossz mozgásokat eredményezhet. Ennek megoldására szokás iteratív optikai áramlást alkalmazni. Ennek a megoldásnak a lényege, hogy az első optikai áramlás meghatározása után az eredeti képet a kiszámolt mozgásvektorok alapján transzformáljuk, majd az így kapott köztes kép és második kép között újra elvégzünk egy optikai áramlás becslést. Ezt a két lépést addig ismételjük, amíg a kapott relatív elmozdulás számottevő. Az algoritmus végén köztes mozgásokat összegezve pedig megkapjuk a tényleges elmozdulást.



4.6. ábra. *Az iteratív optikai áramlás elve: az első iterációval (bal) a képet már közelebb hoztuk az elmozdult változathoz, így a második körben (jobb) már jó becslést kapunk.*

Az iteratív optikai áramlást gyakran szokás piramis módszerrel együttesen alkalmazni. Az eljárás első lépéseként egy durva-finom felbontású képpiramist hozunk létre átméretezések segítségével. Ezt követően iteratív módszerrel megbecsüljük az áramlást a legkisebb felbontású képen. A kapott áramlás valamelyest pontatlan lesz, azonban a nagy mozgások könnyedén meghatározhatók rajta, hiszen ezek az alacsonyabb felbontás miatt ezen a képen pixelben mérve egy nagyságrenddel kisebbek. Ezt követően az iteratív módszert az egyes nagyobb felbontású képen folytatjuk, úgy, hogy a már meglévő áramlás képet használjuk kezdeti értékként. Ily módon minden szinten az előző körben kiszámolt durva áramlás képet finomítjuk tovább.



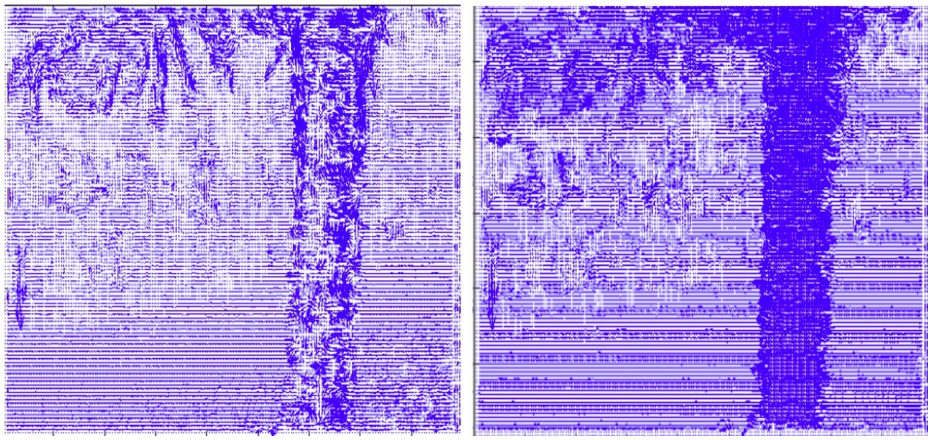
4.7. ábra. *A képpiramis.*

Alkalmazás: Az optikai áramlást számos területen szokás alkalmazni, amelyek közül az egyik jelentős a mozgókamerás 3D-rekonstrukciók végzése. Az algoritmus használatával lehetőség nyílik arra, hogy egy arra egyébként nem alkalmas, egyszerű kamera segítségével térbeli felvételeket készítsünk. Ehhez a kamerával egy, az objektumot körbejáró videót készítünk, majd az optikai áramlás segítségével az egyes pixelek elmozdulását kiszámoljuk. Az elmozdulás mértékéből a kamerától való távolságra tudunk következtetni (a távoli objektumról származó pixelek kisebb sebességgel mozognak a képen).

További érdekes felhasználás a különböző mozgásesemények osztályozása. A képen mozgó objektumokat ugyanis az áramlás képe alapján beoszthatjuk különböző irányokba haladó, forgó, illetve kamerához közeledő vagy attól távolodó objektumokba. Ez rendkívül hasznos, ha szeretnénk észlelni vagy elkerülni a kamerának helyet adó berendezéshez való közeledést vagy az azzal való ütközést.

4.4. Élkeresés

A képi élek definíció szerint a képen található szomszédos pixelek között végbemenő nagymértékű, egyirányú intenzitásváltozások. Lényeges tulajdonságuk, hogy az intenzitás csak az egyik irány-

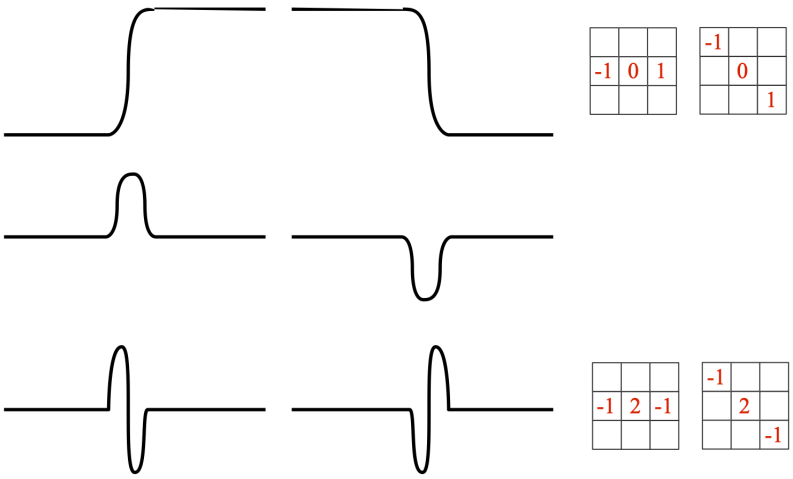


4.8. ábra. *Az optikai áramlás piramis módszerrel (jobb) és anélkül (bal).*

ban változik, míg a másikban konstans, valamint hogy a változás éles, ugrásszerű. A valóságban természetesen a különböző képi hibák, zajok és a véges felbontás miatt a fent leírt ideális élekhez képest a valóságban az átmenet fokozatos, elmosott lesz, valamint lokálisan más irányú változás is elképzelhető.

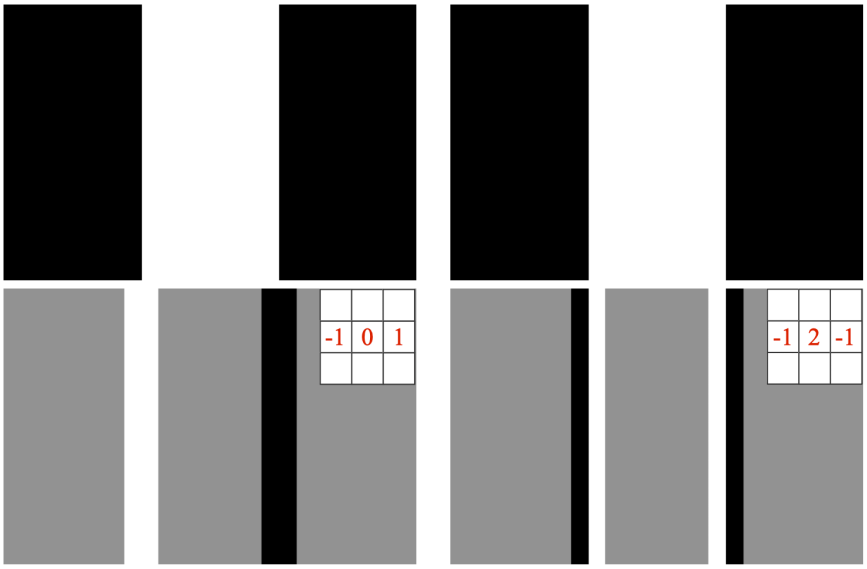
4.4.1. Derivált alapú élkeresés

A képi élek keresésének legegyszerűbb módja a pixelek egyes irányok szerinti deriváltjának számolása, amelyet a konvolúciós szűréshez hasonló elven tehetünk meg numerikusan. A képen végighaladva minden pozícióban kiszámítjuk az adott pixel és a jobb, illetve alsó szomszédja különbségét, megkapva a kép x, illetve y irányú deriváltjait. A kettő négyzetes összegéből megkapható a teljes derivált nagysága, amely az adott pont élszerűségének mértékeként értelmezhető.



4.9. ábra. *A deriváló szűrők alkalmazása egy képre.*

A fent leírt módszer rendkívül gyors és egyszerű, alapvető problémája, hogy a véletlen képi zajok hamis deriváltakat eredményeznek a képen, így a kapott él kép rendkívül zajos lesz. Ez elkerülhető, ha a képet egy Gauss-szűrő segítségével szűrjük, így mérsékelve a zaj hatását. A gyakorlatban azonban az algoritmus gyorsításának érdekében kihasználjuk, hogy mind a deriváló, mind a Gauss-szűrők lineáris műveletek, ezért összevonhatók egyetlen műveletté. Így a két szűrő egymás utáni alkalmazása helyett csak egyetlen szűrést végzünk a Gauss-szűrő deriváltja által meghatározott konvolúciós szűrővel, amely az előző módszerrel megegyező eredményt ad.



4.10. ábra. *Az első (bal) és a második (jobb) deriváltak egy él esetében. Látható, hogy a második derivált alkalmazásakor az él a nullátmenetnél található.*

A Gauss-szűrő deriváltja mellett elterjedtek továbbá további konvolúciós éldetektáló szűrők, amelyek hasonló elven működnek. Ezek közül a legismertebbek a Prewitt- és a Sobel-operátorok, amelyek irányfüggő éldetektorok. Alkalmazásuk esetén, amennyiben bármilyen irányultságú éleket szeretnénk detektálni, akkor az adott operátor mindkét változatát futtatni kell a képen. A két operátor közti alapvető különbség, hogy a Sobel operátor az élre ellentétes irányban simítást is végez, így a zajokra kevésbé érzékeny.

1	0	-1
1	0	-1
1	0	-1

1	0	-1
2	0	-2
1	0	-1

4.11. ábra. *A Prewitt (bal) és a Sobel (jobb) operátorok függőleges élekre.*

Az első deriválton alapuló szűrők egyik legnagyobb hátránya, hogy a simítás miatt az él homályosan, elkenődve fog látszani, így annak pontos lokalizációja nehézkes. Ez a probléma kiküszöbölhető, ha az élképet még egyszer deriváljuk, és a deriváltak nullátmenetének pontját keressük meg. Ezt a műveletet természetesen egyetlen lépésben végezzük el egy második derivált konvolúciós szűrő segítségével, amelyet Laplace-szűrőnek nevezünk. A Laplace-szűrőt gyakorta szokták alakja miatt sombrerokalap-szűrőnek is nevezni.

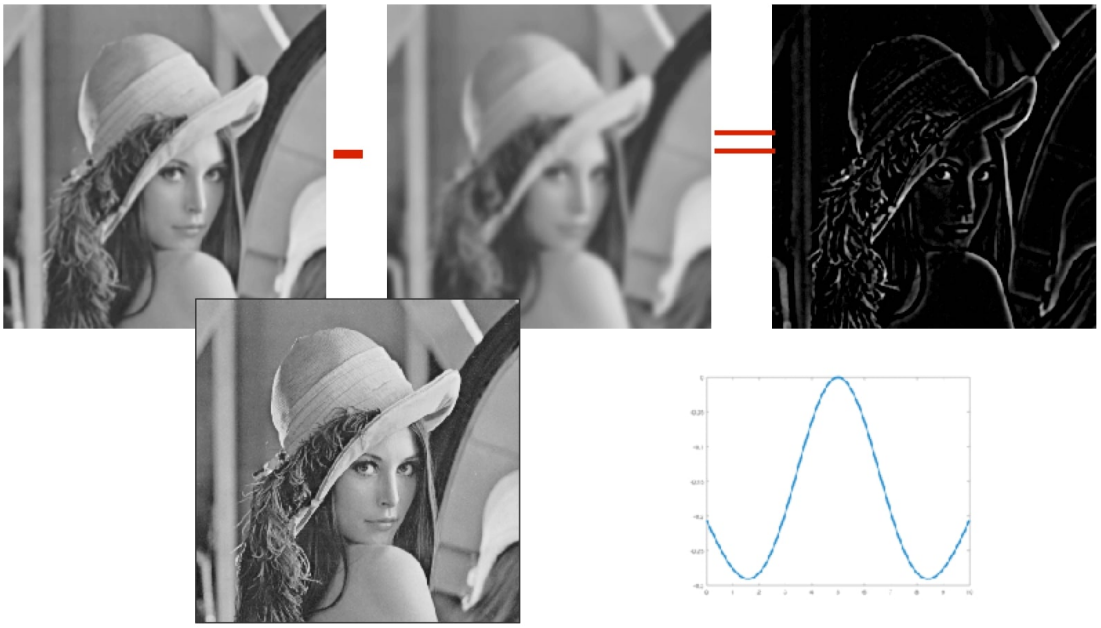
A gyakorlatban a Laplace-szűrőt néha két eltérő szórású Gauss-szűrő különbségével szokták helyettesíteni. Ezt a megoldást DoG-szűrőnek nevezzük (a DoG az angol Difference of Gaussians kifejezés rövidítése), és számos alkalmazásban használatos. Gyakorta előfordul, hogy a képeket egyszerre több, különböző méretű DoG-szűrő segítségével szeretnénk megszűrni. Ekkor bevett szokás a folyamatot úgy gyorsítani, hogy először külön előállítjuk a különböző Gauss-szűrők által simított képeket, majd magukat a szűrt képeket vonjuk ki egymásból. A kapott eredmény az elvégzett műveletek linearitása miatt megegyezik.

Alkalmazás: Az élet számos szituációjában előfordulhat, hogy egy számunkra fontos objektum-

0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

4.12. ábra. *A Laplace szűrő 4 (bal) és 8 (jobb) szomszédos változata.*



4.13. ábra. *A DoG szűrő alkalmazásának elve. Kékkel a DoG szűrő alakja látható 1D esetben.*

ról automatikusan kell felvételt készítenünk. A felvételkészítés automatizálása szükséges lehet, amennyiben nem tudjuk kontrollálni, hogy az objektum mikor és hol jelenik meg a képen, vagy olyan nagy mennyiségű felvételt kell készítenünk, hogy a folyamat automatizálására kell szorítkoznunk. Ebben az esetben azonban nem tudjuk garantálni, hogy a kép jól fókuszált lesz, aminek következményeképp a számunkra releváns objektum részletei elmosódottak lesznek, ami a további feldolgozást ellehetetlenítheti.

Ennek elkerülésére használhatunk automatikus fókuszérzékelést, amelynek lényege, hogy – a kép fókuszáltságának mértékét a képből meghatározva – a kamera fókuszát addig módosítjuk, amíg a mérték maximumát produkáló fókuszbeállítást meg nem találtuk. Mivel egy homályos képen a legtöbb képi él elkenődik, ezért a hozzájuk tartozó képi gradiensek nagysága is kisebb lesz. Innen kaphatunk egy kézenfekvő mértéket: minél több olyan pixel van a képen, amelyek helyén a derivált nagysága egy bizonyos küszöbértéknél nagyobb, a kép annál jobb fókuszú. Fontos megjegyezni, hogy ezzel a módszerrel csak azonos tartalmú képek hasonlíthatók össze.

4.4.2. Canny algoritmus

Az eddigi éldetektáló algoritmusok különböző deriváltszámítási módszerekre alapultak, amelyek során megkaptuk minden pixel élszerűségének mértékét. Innentől kezdve azonban a mi feladatunk

tervezőként eldönteni, hogy pontosan mekkora határérték felett tekintünk egy képpontot élnek. Ha ezt a határértéket túlságosan magasra szabjuk meg, akkor előfordulhat, hogy a valós élek kevésbé kontrasztos részeit nem fogjuk tudni detektálni, ha viszont túl alacsonyra tesszük, akkor rengeteg hamis élt fogunk kapni a zajnak és egyéb hatásoknak köszönhetően. A cél persze a kettő közötti kompromisszum megtalálása, azonban ez sem lesz tökéletes.

Ezt a dilemmát igyekszik kezelni az éldetektáló algoritmusok state-of-the-art módszere, a Canny-algoritmus. A Canny-eljárás egy többlépcsős algoritmus, amelynek első lépése, hogy egyszerű deriváló szűrők segítségével kiszámoljuk a képen a vízszintes és a függőleges irányultságú deriváltakat. Ezt követően a kétirányú deriváltakból minden képpontban meghatározzuk a képi gradiens (a legnagyobb intenzitásváltozás iránya) nagyságát és irányát.

Ezt követően minden egyes élszerű pontból kiindulva, a gradiens irányát követve meghatározzuk a legnagyobb derivált értékkel rendelkező pontot. Az ilyen pontokat meghagyjuk, míg a náluk kisebb derivált értékkel rendelkező szomszédjaikat nullába állítjuk. Ezzel a módszerrel elérjük, hogy a deriváló szűrő használata után kapott homályos élkép pontosan olyan éles legyen, mintha második deriváltat használtunk volna.

Ezt követően a gradiensnagyságokat tartalmazó képet küszöbözzük, azonban egy helyett két különböző küszöbértéket használunk (és így két különböző bináris kép keletkezik). A kisebb küszöbértékkel készített bináris képen nagy valószínűséggel megmarad az összes valódi élpont, azonban számos nem valódi élhez tartozó zaj is keletkezni fog. A nagyobb küszöbértékkel készített bináris képen a valódi élekből biztosan látszani fog valahány képpont, azonban hiányosak lesznek. Cserébe természetesen csak minimális számú zaj fog keletkezni.

4.14. ábra. *A Canny algoritmus által használt élképek.*

A Canny-algoritmus fő erénye, hogy utolsó lépésben e két bináris kép felhasználásával képes egy mindkettőnél lényegesen jobb minőségű élképet létrehozni. Ezt úgy teszi, hogy a nagyobb küszöbértékkel rendelkező, hiányos élképhez hozzáveszi a másik élképről azokat az élpontokat, amelyek szomszédosak a hiányos képen is megtalálható élpontokkal. Ezt iteratívan addig végzi, amíg már nem tud több élpontot hozzáadni a hiányos élképhez. Ennek eredményeképp a Canny-algoritmus fel tudja használni a megengedőbb küszöbértékkel készült élképet arra, hogy a szigorúbb képen keletkező hiányokat betöltse, a valódi élekhez nem kapcsolódó zajokat azonban nem veszi át.

4.5. Hough-transzformáció

Az élkeresés során rendkívül gyakran előfordul, hogy különféle egyszerű alakzatok (téglalap, kör) határvonalait keressük, amely esetben célszerű lehet a megtalált élpontokra egy egyenes modellt illeszteni, így a képen megtalált egyenesszerű elrendezésben található pontokat egy paraméteres modellel leírhatjuk, amely az alakzatok detektálását rendkívül megkönnyíti. A probléma nehézsége, hogy természetesen a képen talált élpontok csak egy része fog egyenesekre illeszkedni, és azok közül is számos pont külön-külön egyenesre illeszkedik, így egyszerre kell azt meghatároznunk, hogy mely

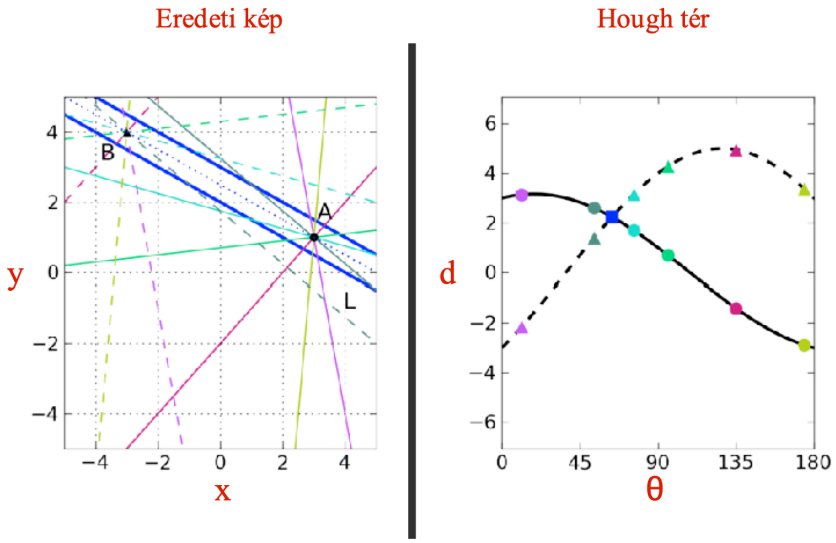
pontok illeszkednek egy egyenesre, és hogy milyen paraméterek írják le ezt az egyenest. Ha a két kérdés közül bármelyikre tudnánk a választ, a probléma megoldása triviális volna.

Erre a problémára az egyik legnépszerűbb algoritmus a Hough-transzformációra épülő alakzatdetektálás. A Hough-transzformáció egy koordinátatranszformáció, amely a kép pontjait a megszokott képkoordinátarendszerből (x, y) az egyenesek paraméterei (r, θ) által kifeszített térbe viszi át. Ebben a térben egy egyenest egyetlen pont (számpár) ír le, amelynek egyik eleme az origóból az egyenesre állított merőleges szakasz hossza (r), a másik eleme pedig ennek a szakasznak az x tengellyel bezárt szöge (θ).

Érdekesebb azonban, hogy a kép egyes pontjai hogyan képződnek le a Hough-térbe. Ezt a leképzést a Hough-transzformáció során úgy végezzük el, hogy az (r, θ) által kifeszített Hough-térben egy képpont képe az összes olyan egyenes lesz, amelyre az adott pont illeszkedik. Habár egy adott pont végtelen számú egyenesre illeszkedik, mégsem illik rá az összes létező egyenesre, így egy konkrét (x, y) pont képét a Hough-térben az alábbi görbe adja meg:

$$x\ cos\theta + y\ sin\theta = r \tag{4.11}$$

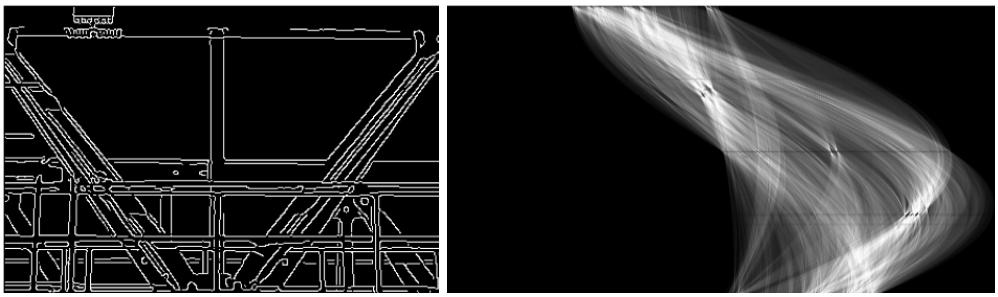
Vagyis egy képpont képe a Hough-térben egy szinuszgörbe lesz. Ha a Hough-térben két pont görbéje metszi egymást, akkor az azt jelenti, hogy mindkét pont ráillik arra az egyenesre, amelyiket a két görbe metszéspontja ír le (emlékezzünk: a Hough-térben egy pont egy egyenest ír le).



4.15. ábra. *A Hough transzformáció elve.*

Ebből az összefüggésből egy egyszerű következtetés vonható le: ha a Hough-térben sok olyan görbénk van, amelyek megközelítőleg egy pontban metszik egymást, akkor ez azt jelenti, hogy a képtérben sok olyan pont van, amelyek ráillenek ugyanarra az egyenesre. Ebből kiindulva a Hough transzformációs egyenesdetektálás algoritmusa könnyedén adja magát: először transzformáljuk az összes élpontot a Hough-térbe, majd megkeressük azt az egyenest, amelyiken a legtöbb ráillő élpontunk található. Ezt követően ezeket az élpontokat töröljük a Hough-térből, majd újra megkeressük a legtöbb pontra illeszkedő egyenest. Ezt addig ismételjük, amíg találunk olyan egyenest, amelyre legalább egy küszöbértéknyi pont illeszkedik.

Fontos megjegyezni, hogy a Hough-transzformációt nemcsak egyenesek, hanem bármilyen egyszerű, paraméterek által leírható formára el lehet végezni. Különböző típusú Hough-transzformációkat alkalmazva tehát különböző egyszerű alakzatokat tudunk detektálni. Egyenesek mellett gyakorta alkalmazzák a Hough-transzformációt körök, illetve ellipszisek detektálására is. Érdemes megemlíteni az általánosított Hough-transzformációt, amelynek segítségével általános formák is észlelhetők.



4.16. ábra. *Egy bináris kép és annak Hough transzformáltja.*

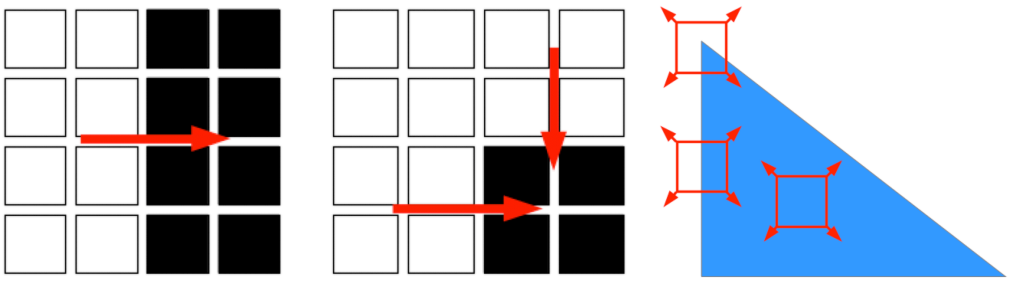
Alkalmazás: A Hough-transzformációt előszeretettel alkalmazzák arra, hogy egyenes szakaszokból álló egyszerű formákat detektáljanak a képen. Ez a módszer könnyedén felhasználható például igazolványkártyák automatikus felismerésére, amely az okos közigazgatás egyik legalapvetőbb feladata. Egy igazolványról készült képen a kártya szélei általában markáns, jól kivehető, egyenes éleket okoznak, amelyeket például a Canny-eljárás könnyedén észlel. Ezt követően a Hough-transzformáció segítségével egyenesekké konvertáljuk ezeket, majd megkeressük az ezek által meghatározott, téglalap alakú objektumot.

4.6. Sarkok

Az első tárgyalt képjellemzők a képi élek voltak, amelyek egyszerű, gyorsan kinyerhető, ellenben mérsékelten robusztus leírói a képeken található objektumoknak. Az élek egyik legfontosabb hiányossága, hogy lokálisan csak egy irányban van változás a képen, így ha az él erre az irányra merőlegesen mozdul el a képen, akkor ezt lehetetlen követni.

Erre a problémára ad megoldást a jelen fejezetben tárgyalt képjellemző, vagyis a képi sarok. Képi sarkoknak nevezünk olyan pontokat a képsíkon, amelyekből minden irányba kiindulva a kép intenzitásértékében számottevő változást figyelhetünk meg. Ebből a definícióból következik, hogy bármerre is mozduljon el a képen a sarkot tartalmazó objektum, a sarok elmozdulását követni tudjuk, így ezt a számunkra valamilyen szempontból releváns objektumot is követhetjük, detektálhatjuk.

Képi sarkok detektálására az egyik legelterjedtebb módszer a Kanade–Lucas–Tomasi-, vagyis a KLT-sarokdetektor. E detektor működésének alapelve, hogy mivel olyan képrészleteket keresünk, amelyek minden irányban jelentősen változnak, ezért egyszerűen az éppen vizsgált képpont egy adott környezetét minden irányban elmozgatjuk a képen, és az elmozgatott és az adott helyen található eredeti képrészletek között négyzetes eltérést számolunk. Ha ez az eltérés minden irányban nagy, akkor sarokszerű képrészletet találtunk.



4.17. ábra. *Az ideális képi él (bal), sarok (közép) és a sarokdetektálás elve (jobb).*

4.6.1. Lokális struktúramátrix

A KLT-detektor a gyakorlatban természetesen nem így dolgozik, mivel e művelet elvégzése minden képpontban rendkívül költséges volna. Ehelyett a KLT-detektor kiszámolja a kép x és y irányú deriváltjait, majd minden képpont esetében annak az $n \times n$-es környezetében található deriváltakat egy $n^2 \times 2$-es mátrixba rendezi. Az egyes pixelek irányába kapott elmozdulás hibája ugyanis felírható az alábbi módon:

$$\begin{pmatrix} I_{x1} & I_{y1} \\ I_{x2} & I_{y2} \\ \vdots & \vdots \\ I_{xn} & I_{yn} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{pmatrix} \tag{4.12}$$

Innen a teljes hiba:

$$||E|\,|^2 = \vec{u}^T I^T I \vec{u} = \vec{u}^T H \vec{u} \tag{4.13}$$

Ahol I_{x1} és I_{y1} a környezet első pixelének (a végeredmény szempontjából a pixelek sorrendje irreleváns) x és y irányú deriváltjai, H pedig az adott képpont úgynevezett lokális struktúramátrixa. Fontos megjegyezni, hogy amennyiben feltételezzük, hogy a képen az egyes deriváltak várható értéke nulla (ami egy teljesen megalapozott feltételezés, hiszen a deriváltak pont ugyanolyan valószínűséggel negatívak, mint pozitívak), akkor a lokális struktúramátrix a képrészlet deriváltjainak kovarianciamátrixával arányos.

4.6.2. KLT, Harris

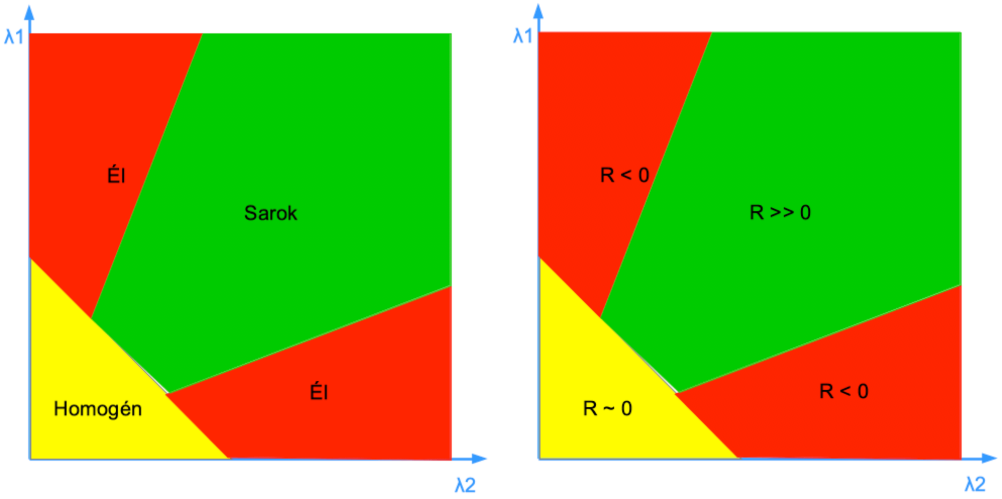
Fontos azonosság, hogy a lokális struktúramátrix két sajátvektora azt a két irányt fogja meghatározni a képsíkon, amely irányokba a kép a leginkább, illetve a legkevésbé változik. E legnagyobb és legkisebb változások mértékét pedig az egyes sajátvektorokhoz tartozó sajátértékek (λ_1, λ_2) adják meg. Ez az összefüggés rendkívül jó módszert kínál nekünk a sarkok detektálására: Azok a pontok, ahol a legkisebb változás is még elég nagy (vagyis a lokális struktúramátrix legkisebb sajátértéke is nagy) sarokpontoknak tekinthetők.

A KLT-detektor működése során a kép minden eleméhez elvégzi a lokális struktúramátrix számítását, és annak kisebbik sajátértéke alapján hozzárendel egy saroksági mércét, amely alapján egy pixel sarokpont volta megállapítható. Fontos megjegyezni, hogy a sarokpont környékén található pixelek sarokságértéke mind nagy lesz, hiszen a képi sarkok nem tökéletesen pontszerű jelenségek. Egyszerű megoldás lehet minden esetben a sarokságértékek lokális maximumait tekinteni a sarokpont helyzetének.

A gyakorlatban szokás a KLT-detektor helyett egy másik eljárást, a Harris-detektort használni, amely szintén a lokális struktúramátrixot használja fel a sarokpontok detektálására, a saroksági mértéket azonban az alábbi képlet alapján határozza meg:

$$\begin{gathered} R = det(H) - k * tr(H)^2 \\ k \in [0.04 - 0.06] \end{gathered} \tag{4.14}$$

Ennek a számolási módszernek hatalmas előnye, hogy lényegesen gyorsabb a sajátértékek meghatározásánál. A Harris-sarokság mellesleg alkalmas élszerű képrészletek detektálására is, ugyanis ilyen jellegű képrészletek esetén a sarokság értéke egy nagy negatív szám lesz. Lényeges további különbség a két detektor között, hogy habár lassabb, a KLT-detektor eredménye közelebb áll az emberi érzékeléshez.



4.18. ábra. *A KLT (bal) és a Harris (jobb) sarokság értékek osztályozási elve.*

4.7. Invarianciák

Képjellemzők tárgyalásánál rendkívül fontos megemlíteni az e jellemzőkkel szembe állított robusztussági követelményeket. Célunk ezekkel a jellemzőkkel az, hogy ha ugyanazt az objektumot több képen is látjuk, de esetleg a két kép között különböző transzformációk történtek, akkor e transzformációk közül a képjellemzők minél többre legyenek invariánsak, mivel így ugyanazt a tárgyat különböző képeken is könnyen észlelhetjük.

A legalapvetőbb képtranszformáció az intenzitásváltozás, amely leggyakrabban a megvilágítás megváltozásának köszönhető. Ennek két típusa létezik. Az egyik az additív változás, amely során a képpontok intenzitásértékéhez egy konstans adódik hozzá. Másik típusa a multiplikatív változás, amelynek során a képpontok intenzitása egy konstanssal szorzódik. További gyakori transzformáció a forgatás, valamint az objektum skálájának változása, amely a különböző nézőpontból és távolságból készült felvételek miatt keletkezik. A nézőpontváltozás okoz még perspektív torzítást is a képeken, amely legjobban a korábban párhuzamos egyenesek összefutásából figyelhető meg.

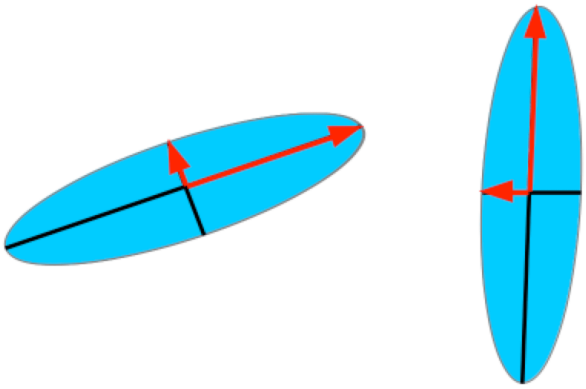
A KLT-, illetve a Harris-operátorok e transzformációk közül sajnálatos módon csak kettőre invariánsak: egyrészt az additív intenzitásváltozásra, aminek az az oka, hogy a kép deriváltjai a konstans tagot kiejtik, valamint az elforgatásra, mivelhogy az elforgatás egy mátrix sajátértékeit nem befolyásolja. Multiplikatív intenzitásváltozás esetén a deriváltak is konstanssal szorzódnak, így a sarokságértékek eszerint fognak megváltozni. A skálázás hatásaként az egyik képen sarokszerűnek látszó objektum felnagyítva egy lekerekített élszerű objektumba mehet át, így a skálázás rendkívüli módon befolyásolja a sarokdetektálás eredményét.

4.8. Komplex képjellemzők

Ezt a problémát igyekszik kezelni a skálainvariáns képjellemző transzformáció, vagyis a SIFT-algoritmus (Scale Invariant Feature Transform), amely a perspektív torzítás kivételével minden transzformációra invariáns, így robusztus lokális régióleírót produkál, amelyet széles körben alkalmaznak. Alapelve, hogy sarokszerű pontokat keres a képen, azonban ezekhez nem egyetlen mértéket, hanem a sarokpontok lokális környezetét invariáns módon leíró kódot készít, amelynek segítségével más képeken megtalált jellemzőkkel összevethető, párosítható.

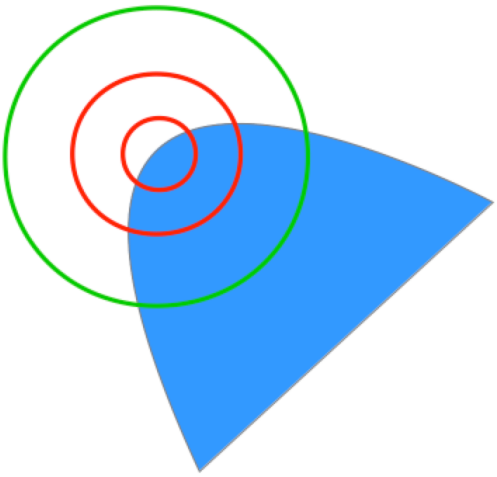
4.8.1. SIFT detektor

A SIFT-algoritmus első számú alapelve, hogy a sarokdetektálást nem egy, hanem több skálafaktor mentén végzi el, és minden jellemzőhöz egy skálaváltozót rendel hozzá, amelyet a leírókód készí-



4.19. ábra. *A forgatás hatása a sajátértékekre.*

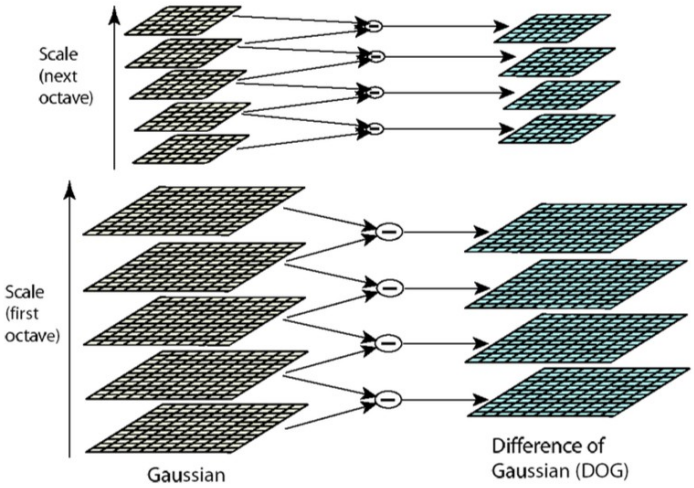
tésekor felhasznál, a skálainvarianciát ilyen módon biztosítva. A sarokdetektálást a módszer a 3. fejezetben említett DoG-szűrők (Difference of Gaussians) segítségével végzi el. Korábban ezt a szűrőfajtát éldetektorként ismertük meg, a DoG-szűrő válasza azonban impulzusszerű kitüremkedések és bemélyedések esetén lesz a lehető legnagyobb. Fontos megjegyezni, hogy ha egy adott impulzusszerű képrészleten különböző méretű DoG-szűrőket futtatunk végig, akkor annak a szűrőnek lesz a lehető legnagyobb a válasza, amelynek a mérete a leginkább egybeesik az impulzus szélességével.



4.20. ábra. *Egy képjellemző (kék) sarokszerűsége több skála mentén vizsgálva.*

A SIFT-algoritmus első lépéseként számos, különböző méretű DoG-szűrőt futtat végig a képen, amelyek válaszát eltárolja. Ezt követően megkeresi szűrőválaszok lokális maximumait, de nem csak a kép x és y koordinátája, hanem a szűrők mérete szerint is. A legnagyobb válasszal rendelkező síkbeli koordináta lesz a sarokpont helyzete, a maximális válaszú szűrő mérete pedig az ehhez hozzárendelt skálafaktor. Érdemes tudni, hogy a SIFT nem elégszik meg a diszkrét pixelek és szűrőméretek által kvantált maximumpozícióval, hanem a válaszértékekre lokálisan egy polinomot illeszt, és ennek a maximumhelyét keresi meg. Ezzel a módszerrel a sarokpont helyzetét szubpixeles (vagyis pixelméret alatti) pontossággal képes meghatározni, amely számos alkalmazás esetében követelmény lehet.

Érdemes megjegyezni, hogy az eljárás a szűrés folyamatát két módon is gyorsítja: egyrészt a DoG-szűrők futtatása helyett az algoritmus különböző méretű Gauss-szűrőket futtat a képeken, majd ezeket vonja ki egymásból, így gyorsítva a működést. Másrészt, amikor az éppen aktuális Gauss-szűrő mérete eléri az eredeti kétszeresét, akkor inkább az eredeti szűrőt futtatja a kép feleakkora felbontású változatán, így ugyanazt az eredményt kapja hozzávetőlegesen negyedannyi számítás segítségével.

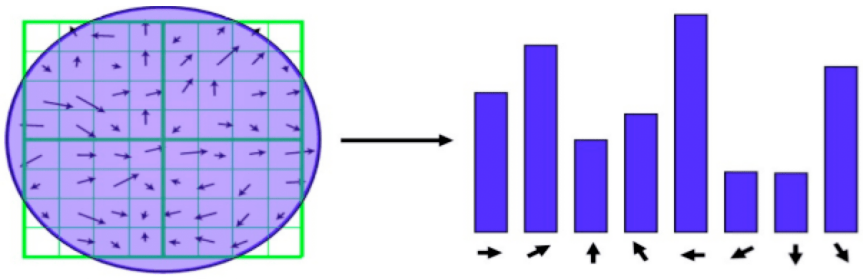


4.21. ábra. *A SIFT detektor szűrési sémája.*

4.8.2. SIFT leíró

A SIFT-algoritmus második lépése a megtalált sarokponthoz tartozó leírókód generálása. A SIFT minden sarokponthoz egy 128 számból álló kódot generál, amely a sarokpont 16×16 pixeles környezetének kinézetét írja le. Ezt a 16×16 pixeles környezetet mindig az adott sarokpont skálája szerint átméretezett képből vesszük, így biztosíthatjuk a leírókód skálainvarianciáját. Mivel a leírókódot nem közvetlenül a környezet intenzitásértékeiből, hanem annak gradienseiből (az x és y irányú deriváltak által alkotott kétdimenziós vektorból) készítjük, így a KLT-eljáráshoz hasonlóan az additív intenzitásváltozásra való invarianciákat is biztosítjuk.

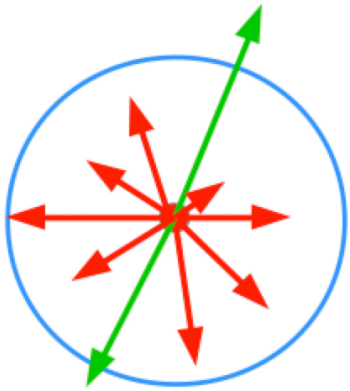
A SIFT-algoritmus egyik legforradalmibb ötlete az elforgatásinvariancia biztosítása. Ennek elvégzéséhez a pont környezetének gradienseihez hisztogramot készítünk, amely azt ábrázolja, hogy az egyes irányokba mekkora gradiensek találhatók ebben a környezetben. A gradienshisztogram készítése során a kép gradienseit 36 irány közül osztjuk be valamelyikbe, így a kört 10 fokos intervallumokra osztjuk. Az intervallumhatárok felé mutató gradienseket egyenlően elosztjuk a szomszédos rekeszek között. Fontos részlet, hogy a sarokponttól távoli képpontok gradienseit kisebb súllyal vesszük figyelembe.



4.22. ábra. *A gradienshisztogram konstruálása (a példaként szereplő hisztogram 8 elemű).*

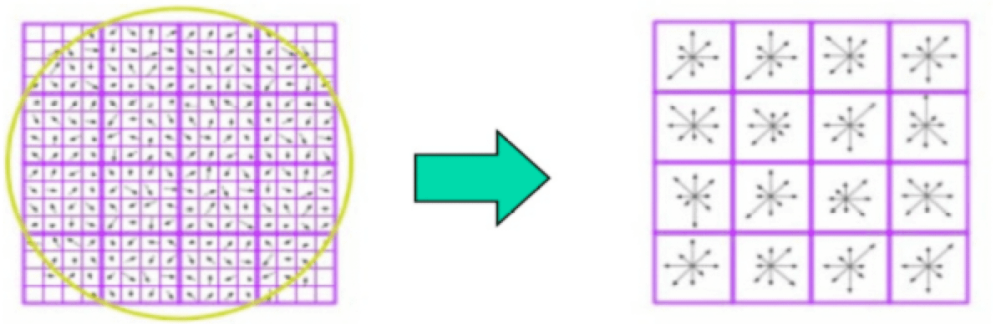
A kapott gradienshisztogram-maximum helyét (vagyis azt az irányt, amelybe a leginkább változik a kép ebben a környezetben) megkeressük, és azt az irányt tesszük meg az adott sarokpont orientációjának. A képet ezt követően úgy forgatjuk el, hogy a sarokpont orientációja egy előre meghatározott irányba (például függőlegesen felfelé) mutasson, és a végső leírókódot az elforgatott (és korábban átskálázott) képen található 16×16 pixeles környezetből készítjük, így biztosítva annak invarianciáját az elforgatásra. Fontos megjegyezni, hogy ha a gradienshisztogram maximális értéke nem egyértelmű, akkor minden, a maximálishoz közel lévő orientációhoz készítünk egy külön leírókódot.

A leírókód készítéséhez a felhasználandó 16×16 pixeles környezetet 16 darab 4×4 pixeles részre



4.23. ábra. *A SIFT orientáció meghatározása (nincs mind a 36 elem ábrázolva).*

osztjuk, és ezekhez külön-külön, a fent leírt módon gradienshisztogramokat készítünk. Az egyetlen különbség, hogy az egyes gradienshisztogramok csak 8 és nem 36 rekeszből állnak, így a felbontásuk 45 fok. Az így kapott hisztogramok értékeit egymás után egy vektorba írjuk, így összesen 128 számot kapunk. Ezt a kapott vektort normáljuk úgy, hogy a benne szereplő számok négyzetösszege egy legyen. Ezzel a normálással elérjük, hogy a multiplikatív intenzitásváltozás, amely a gradienseket is konstanssal szorozza, a végső leírókódot ne tudja befolyásolni.



4.24. ábra. *A végső leíró vektor konstruálása.*

Alkalmazás: A lokális képjellemzőknek számos alkalmazási területe létezik. Használatosak például képek illesztésére, amely panorámakészítésnél vagy 3D-látásnál rendkívül fontos. Az egyik legkézenfekvőbb alkalmazás azonban merev objektumok felismerése referenciakép alapján. Ebben az esetben olyan tárgyakat kívánunk felismerni és lokalizálni, amelyek nem deformálódnak, valamint nem létezik túlságosan nagyszámú variáció belőlük. Ekkor a lokális gradienshisztogram alapú képjellemzőket a referenciaképen megkeressük, majd az aktuális képen található jellemzőket ezekkel összevetjük, a kódok hasonlósága alapján párosítjuk. Nagyszámú egyezés esetén a keresett objektumot megtaláltuk.

Ezt a módszert gyakran alkalmazzák akkor, ha olyan objektumokat kell megkeresni, amiket a rendszer tervezője már nem tud befolyásolni, vagyis nem tudjuk könnyebben felismerhetővé tenni. Gyakran alkalmaznak ilyen módszereket valós tárgyakat is felhasználó virtuális- vagy kiterjesztettvalóságrendszerek, valamint különböző kameraalapú megfigyelő- és követőrendszerek (például tér- és forgalommegfigyelő kamerás rendszerek).

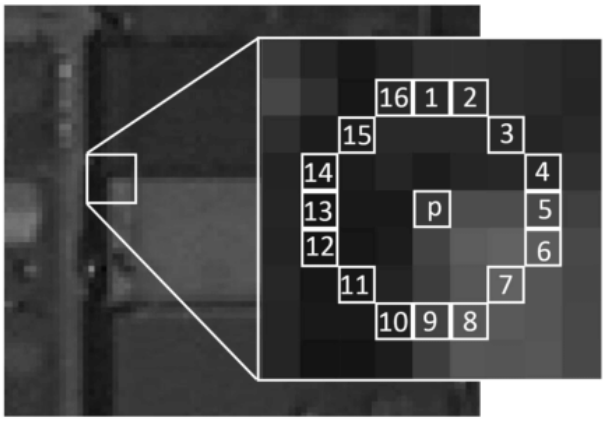
4.8.3. ORB detektor

A SIFT-eljárás végére egy olyan jellemződetektálási és -leírási módszert kaptunk, amely a fent említett transzformációkra invariáns. A módszer hátránya, hogy rendkívül számításigényes, mo-

dern asztali számítógépeken sem mindig képes valós időben futni (a szabadalom szerencsére 2020-ban lejárt). Az algoritmus publikálása óta azonban készült számos más eljárás, amelyek mind a SIFT alapötletére építenek, de lényegesen kevesebb számítást igényelnek. Ezek közül kiemelendő a SURF-módszer, amely hasonló robusztusság mellett kiválóan párhuzamosítható grafikus kártyák segítségével, valamint az ORB, amely az invarianciák megtartása mellett tud valós időben futni.

Az ORB (Oriented FAST and Rotated BRIEF) algoritmus két másik módszer kombinálásából és kiegészítéséből adódik. Az ORB algoritmus a FAST elnevezésű gyors sarokdetektort alkalmazza a kulcspontok detektálására. A FAST eljárás lényege, hogy az éppen vizsgált pont környezetében egy kört definiál és az így meghatározott körön elhelyezkedő pixeleket választja ki a sarokság vizsgálatához. Ezt követően az algoritmus összeszámolja, hogy az így kiválasztott 16 pixel közül mennyinek különbözik az intenzitása a központi pixeltől egy adott küszöbértéknél jobban. Ahol ez a szám nagy, ott sarokszerű képpontot detektáltunk.

Érdemes megjegyezni, hogy ha a különböző pixelek számára adott külszöbérték 12, vagy nagyobb, akkor a nem sarokszerű pontok rendkívül gyorsan kizárhatók a négy égtáj irányába való ellenőrzéssel. Ellenkező esetben is megoldható a nem sarokszerű pixelek gyors kizárása, ehhez azonban egy döntési fát kell alkalmaznunk.



4.25. ábra. *A FAST detektor működési elve.*

Fontos megjegyezni, hogy a skálainvariancia biztosításának érdekében itt is egy képpiramist használunk arra, hogy a sarokdetektálást több skálafaktor mellett el tudjuk végezni. A SIFT eljáráshoz hasonlóan a végső leírót a sarokponthoz kiválasztott skálafaktorhoz tartozó kép felhasználásával készítjük. Fontos még megjegyezni azt is, hogy a FAST detektor hajlamos élszerű képpontokat is megtalálni, így a detektált kulcspontokat érdemes a KLT sarokság segítségével szűrni. Ebben az esetben azonban már csak néhány pontra kell a sarokságot kiszámolni, ami elhanyagolható mennyiségű számítást von maga után.

A FAST detektor egyik lényeges limitációja, hogy nem tud orientációt rendelni a megtalált kulcspontokhoz. Ennek megoldására külön orientáció mércét kell definiálni. Az FAST algoritmus ezt úgy oldja meg, hogy a kulcspont környezetében kiszámolja a pixelek tömegközéppontját, ahol az egyes pixelek relatív tömegét az intenzitásuk határozza meg. Ennek képlete az alábbi:

$$x = \frac{\sum xI(x,y)}{\sum I(x,y)}; \qquad y = \frac{\sum yI(x,y)}{\sum I(x,y)} \tag{4.15}$$

Innen az ORB jellemző orientációja a kulcspontból a tömegközéppontba mutató vektor iránya lesz.

4.8.4. ORB leíró

Az ORB algoritmus a leíró vektor generálásához a BRIEF leíró módszert alkalmazza, melynek alapvető tulajdonsága, hogy a SIFT leíróval szemben nem lebegőpontos számok, hanem bináris

elemek vektorával jellemzi a képrészletet. Ehhez az algoritmus vesz n_d (ez az ORB esetében 256) darab előre definiált pontpárt a kulcspont környezetében, és ezekre egyesével megvizsgálja, hogy a pontpár első tagjának intenzitása nagyobb-e, mint a másikénak. Ily módon a BRIEF leíró alakja a következő:

$$\left[sign(I(x_1^{(1)}) - I(x_2^{(1)})) \quad sign(I(x_1^{(2)}) - I(x_2^{(2)})) \quad \dots \quad sign(I(x_1^{(n_d)}) - I(x_2^{(n_d)}))\right] \tag{4.16}$$

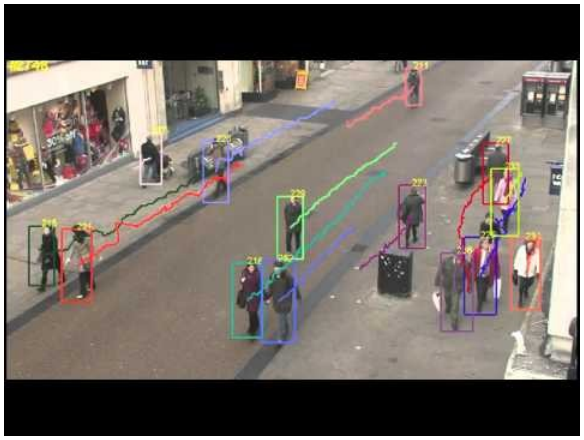
Vizsgáljuk meg a BRIEF leíró invarianciáit. Mivel a leíró egyedül az egyes pixelek intenzitásainak különbségét veszi figyelembe, ezért mind a kétféle intenzitásváltozásra triviálisan invariáns. A skálainvarianciát a SIFT módszertől ellesett többskálás detektálás és leírógenerálás módszerével oldja meg. Problémát jelent azonban az elforgatásra adott invariancia, mivel az előre meghatározott pontpárok egy elforgatott képrészlet esetében más pixelekre fognak esni.

Ez a probléma azonban könnyedén orvosolható volna a képrészlet elforgatásával, ez azonban jelentős számítást igénylő művelet. Ehelyett sokkal egyszerűbb, ha az orientációkat 12 fokonként kvantáljuk (így 30 lehetséges orientációnk lesz), és nem a képrészletet, hanem a kiválasztandó pontpárok koordinátáit forgatjuk el. Mivel a pontpárok koordinátáit előre, az algoritmus megírásakor választjuk ki, ezért ezeket az elforgatott változatokat is előre kiszámolhatjuk, így az algoritmus futásakor már csak az adott képjellemző orientációjának megfelelő listából kell ezeket kiolvasni.

Érdemes megjegyezni, hogy az ORB algoritmus leíróinak összehasonlítását nem célszerű a SIFT esetében alkalmazható négyzetes távolság segítségével elvégezni, ez ugyanis nem feltétlenül ad jó eredményt. Mivel az ORB leíró egy bináris vektor, ezért két ilyen leíró hasonlóságát általában a Hamming-távolság segítségével számítjuk.

4.9. Követés

A számítógépes látás tudományterületén belül külön figyelmet szentelünk annak a meglehetősen gyakori esetnek, amikor a feldolgozás alapja nem állókép, hanem egy videó. A videókon végrehajtandó feladatok hasonlók az állóképeken végzendőkhöz. Végezhetünk például szegmentálást, detektálást, valamint osztályozást is, azonban megjelennek újabb feladatok, például a mozgásérzékelés és -követés, valamint a különböző események felismerése. A korábban tárgyalt képjellemzők különösen alkalmasak a követési feladatok elvégzésére.

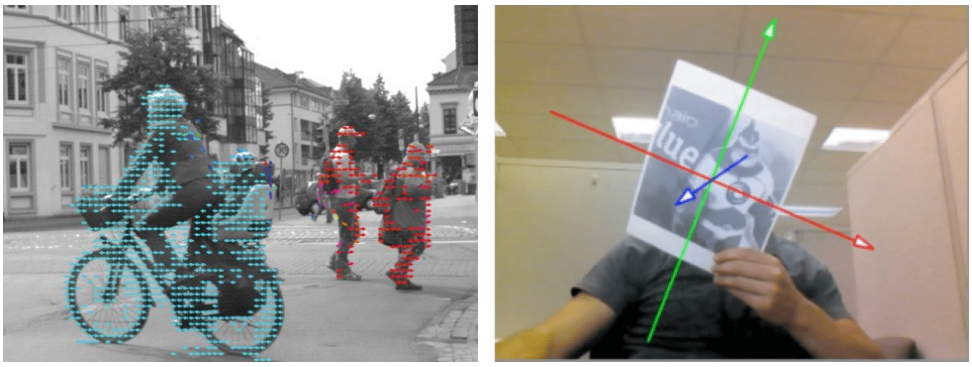


4.26. ábra. *Objektumkövetés.*

Ahhoz, hogy ilyen jellegű bemenetek esetén is tudjunk feldolgozást végezni, először tisztázni kell a videók reprezentációjának módját. Az esetek túlnyomó többségében egy videót egyszerűen állóképek sorozataként értelmezünk, amelyeket a beérkezésük sorrendjében dolgozunk fel. Fontos megjegyezni, hogy az eljárásainkat általában úgy alkotjuk meg, hogy nem használjuk fel az éppen feldolgozandó képkockát követő, jövőbeli képkockák által hordozott információt. Ezt esetleg meg lehet tenni, ha nem valósidejű feldolgozás a célunk, ellenkező esetben viszont erre nincs lehetőség.

Az állóképek sorozataként történő feldolgozás mellett lehetséges még a videót egy alapkép és az azt követő különbségképek sorozataként is értelmezni, de ezt a számítógépes látás során ritkán alkalmazzuk.

Követési módszerek közül megkülönböztethetünk pixel- és objektumszintű követést. Utóbbi esetben fontos az egyes objektumok azonosítása a képkockák között. Ez azonban nehézkes, ugyanis maga az objektum is megváltozhat a két felvétel között, valamint egyéb nehezítő hatások (eltakarás, nemlineáris mozgás, stb.) is tovább nehezíthetik az azonosítást. Pixel szintű követés esetében azonban felhasználhatók a korábbiakban ismertetett jellemző követő módszerek, mint az optikai áramlás, sarokdeketálás, vagy SIFT/SURF/ORB.



4.27. ábra. *Pixelek és objektumok követése.*

4.10. Hidden Markov Model

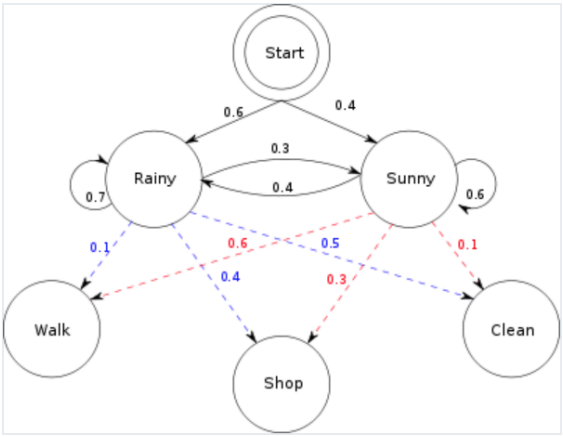
Az optikai áramlás felhasználható arra, hogy egyes objektumok elmozdulását meghatározzuk, úgy, hogy az előző képkockán ismert pozícióból történt relatív elmozdulást határozzuk meg. Célszerű lehet azonban hagyományos azonosításra alapuló objektumkövetés esetén is felhasználni valahogyan az előző ismert pozíciót, és a két információ segítségével valahogyan egy jobb becslést adni. Erre a célra gyakorta használnak rejtett Markov modelleket (HMM).

A rejtett Markov modellek a Markov folyamatokra épülnek. Markov folyamatnak nevezünk egy folyamatot akkor, ha a következő állapot csak a jelenlegi állapottól függ, egyik korábbitól sem (vagyis a folyamat állapotai teljes mértékben kódolják a folyamat teljes történetét). Diszkrét, véges állapottérrel rendelkező Markov folyamatokat könnyedén jellemezhetünk egy állapotgráf segítségével.

A rejtett Markov modellek esetében azonban nem tudjuk a folyamat állapotait közvetlenül megfigyelni, hanem csak azoknak valamilyen következményeit. Erre jó példa, ha van információnk egy ember által elvégzett napi tevékenységekről (séta, vásárlás, takarítás), és ebből szeretnénk az időjárásra (napos, esős) következtetni. Ebben az esetben a Markov modell rejtett állapotai az időjárási állapotok lesznek, a megfigyelések pedig az elvégzett tevékenységek.

Rejtett Markov modellek esetében két tipikus problémát lehet megoldani. Az első, hogy egy adott rejtett állapot sorozat valószínűségét határozzuk meg. Az állapotátmeneti valószínűségek ismeretében ez a feladat triviálisan megoldható. A második, érdekesebb feladat az, ha van egy adott megfigyelés sorozatunk, és meg kéne határozni ennek alapján a legvalószínűbb rejtett állapot sorozatot. Erre is létezik megoldás, a leginkább elterjedt a Viterbi algoritmus.

Objektumkövetés esetén a rejtett állapot az objektum tényleges pozíciója, az állapotátmenetet pedig többféle módon is megválaszthatjuk. Lehetséges egy bizonyos környezetben egyenletes eloszlást választani, azonban célszerű valamilyen (meglehetősen széles) normális eloszlást választani, hiszen a nagyobb mozgások kevésbé valószínűek. A modell megfigyelései az objektum valamely korábban ismertetett eljárás segítségével becsült pozíciója. Feltételezhetjük, hogy az állapot és a megfigyelés közti valószínűséget is valamilyen gauss-szerű eloszlás adja meg. Ebben az esetben célszerű lehet valamilyen vastagabb farokrésszel rendelkező eloszlás (pl.: Student T eloszlás) használata.

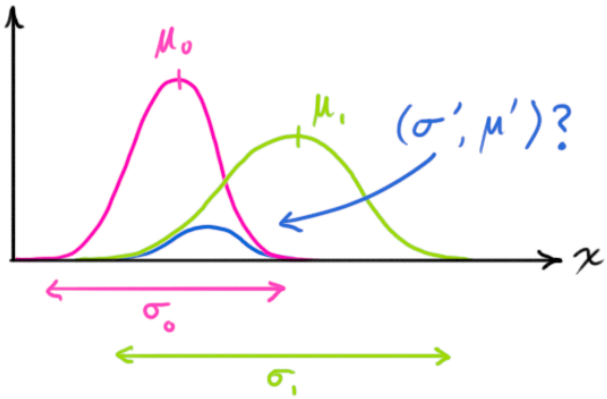


4.28. ábra. *A Rejtett Markov-Modell: ebben a példában a rejtett állapotok az időjárást írják le, a megfigyelések pedig egy adott ember napi tevékenységeit.*

4.11. Kalman-szűrő

Előfordulhat, hogy az objektum pozíciójára nem egy, hanem két különböző forrásból származó becslésünk is van. Mindkét becslésről feltételezzük, hogy gauss eloszlásúak eltérő szórásokkal. Ekkor felmerülhet bennünk az igény, hogy a kettő segítségével meghatározzunk egy kombinált becslést, aminek a szórása kisebb lesz. Ezt a következőképp tehetjük meg:

$$\begin{aligned} \mu &= \frac{\sigma_1^2\mu_0 + \sigma_0^2\mu_1}{\sigma_0^2 + \sigma_1^2} = \mu_0 + k(\mu_1 - \mu_0) \\ \sigma^2 &= \frac{\sigma_0^2\sigma_1^2}{\sigma_0^2 + \sigma_1^2} = \sigma_0^2 - k\sigma_0^2 \\ k &= \frac{\sigma_0^2}{\sigma_0^2 + \sigma_1^2} \end{aligned} \tag{4.17}$$



4.29. ábra. *Gauss eloszlású becslések kombinációja.*

A képlet mögött intuíció, hogy az új várható érték a két becslés várható értékeinek súlyozott átlaga, ahol a súlyok az egyes becslések megbízhatóságával arányosak. A becslések megbízhatósága pedig a szórásukkal fordítottan arányos, ezért cserélődnek meg az indexek a számlálóban. Az eredő becslés megbízhatósága pedig az eredeti megbízhatóságok összege lesz, vagyis a szórást a replusz művelet eredményezi. A fenti képlet többváltozós normális eloszlások esetében is működik, ekkor azonban szórás helyett kovariancia mátrix áll rendelkezésünkre.

$$\begin{aligned} \vec{\mu} &= \vec{\mu}_0 + K(\vec{\mu}_1 - \vec{\mu}_0) \\ \Sigma &= \Sigma_0 - K\Sigma_0 \\ K &= \Sigma_0(\Sigma_0 + \Sigma_1)^{-1} \end{aligned} \tag{4.18}$$

A képletekben megjelenő k és K az úgynevezett Kalman-erősítés.

Kalman-szűrő esetében a két becslésünk közül az egyik mérésből, a másik pedig a rendszer dinamikájának ismerete alapján elvégzett predikcióból származik. Lineáris esetben a rendszer dinamikáját az alábbi állapotegyenlet adja meg:

$$\begin{aligned} x_k &= F_k x_{k-1} + B_k u_k \\ y_k &= H_k x_k \end{aligned} \tag{4.19}$$

Ha ismerjük az előző állapot becslését $N(\hat{x}_{k-1}, \Sigma_{k-1})$, akkor a predikció az alábbi módon adódik:

$$\begin{aligned} \hat{x}_k &= F_k x_{k-1} + B_k u_k \\ \Sigma_k &= F_k \Sigma_{k-1} F_k^T + Q_k \end{aligned} \tag{4.20}$$

Ahol Q_k a predikció bizonytalansága. Mivel gyakorta nem tudjuk a rendszer állapotát közvetlenül mérni, hanem csak a kimenetről, ezért az állapot becsléséből még egy kimeneti becslést is elő kell állítani az alábbi módon:

$$\begin{aligned} \hat{y}_k &= H_k \hat{x}_k \\ \hat{\Sigma}_k &= H_k \Sigma_k H_k^T \end{aligned} \tag{4.21}$$

Ekkor már a kimeneti mérést $N(z_k, R_k)$ és a becslést a fent ismertetett képlet alapján összekombinálhatjuk:

$$\begin{aligned} K_k = H_k \hat{\Sigma}_k H_k^T (H_k \hat{\Sigma}_k H_k^T + R_k)^{-1} \quad &\leftarrow \quad Kalman\ gain \\ H_k x_k = H_k \hat{x}_k + K_k(z_k - H_k \hat{x}_k) \quad &\leftarrow \quad \mu\ kifejezese \\ H_k \Sigma_k H_k^T = H_k \hat{\Sigma}_k H_k^T - K_k H_k \hat{\Sigma}_k H_k^T \quad &\leftarrow \quad \Sigma\ kifejezese \end{aligned} \tag{4.22}$$

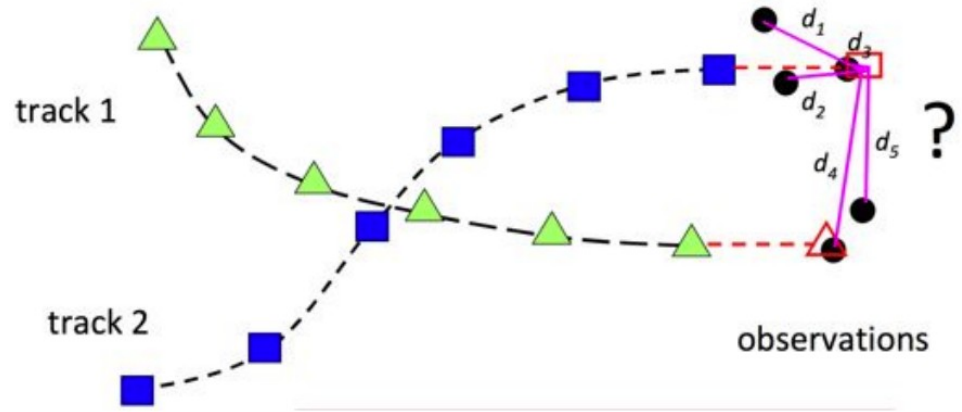
Ennél a pontnál azonban még nem az x állapot, hanem az y kimenet becslését végeztük el. Vegyük észre, hogy H_k inverzével balról szorozva az kiejthető (Az egyik H_k a K kifejezéséből ejthető ki), valamint a kovarianciamátrix egyenletéből H_k^T jobbról is kiejthető. Így a végső becslése:

$$\begin{aligned} \hat{K}_k &= \hat{\Sigma}_k H_k^T (H_k \hat{\Sigma}_k H_k^T + R_k)^{-1} \\ x_k &= \hat{x}_k + \hat{K}_k(z_k - H_k \hat{x}_k) \\ \Sigma_k &= \hat{\Sigma}_k - \hat{K}_k H_k \hat{\Sigma}_k \end{aligned} \tag{4.23}$$

4.12. Több objektum követése

Mindez idáig egyetlen objektum követéséről esett szó. Amennyiben azonban több objektumot is kell követni, akkor könnyedén előfordulhatnak olyan esetek (különösképp, amikor objektumok pályája keresztezi egymást), amikor nem egyértelmű, hogy az egyes detektált objektumokat hogyan érdemes a korábbi képkockán ismert objektumokhoz hozzárendelni. A probléma megfogalmazható egy optimalizálási problémaként az alábbi módon:

$$max_x \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} x_{ij} \quad ahol \quad \begin{cases} \sum_j x_{ij} = 1 & \forall i \\ \sum_i x_{ij} = 1 & \forall j \\ x_{ij} \in \{0, 1\} \end{cases} \tag{4.24}$$



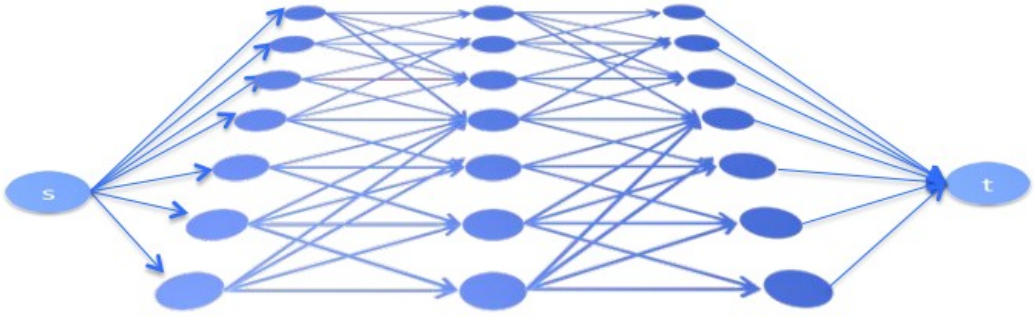
4.30. ábra. *Több objektum követése.*

Ahol w_{ij} az korábbról ismert és az aktuális detektált objektum tulajdonságai közti hibával fordítottan arányos. Az azonosítás elvégzésére használhatunk számos jellemzőt. Ezek közül a legegyszerűbbek a pozíció, a befoglaló téglalap vagy ellipszis alkalmazása. Lehetséges azonban a 3. előadásban megismert lokális képjellemzők, vagy az 5. előadásban ismertetett különböző bináris leíró módszerek használata. Minden esetben definiálhatjuk azonban az affinitást, amely két objektum leírására használt jellemzők közti hasonlóságot írja le:

$$A(x,y) = e^{\frac{-1}{2\sigma^2}||f(x)-f(y)||^2} \tag{4.25}$$

Ahol $f(x)$ az x objektumot leíró jellemző vektor. Ezt a mennyiséget használhatjuk, mint súlyokat a fenti problémában.

Ezt a párosítási problémának is nevezett feladatot megoldhatjuk a Magyar algoritmus segítségével amennyiben csak egy képkockáról van szó. Több képkocka esetén azonban már nem garantált az optimális megoldás. Ebben az esetben az egyes objektumok lehetséges pályái leírhatók egy súlyozott, irányított gráf segítségével. Az optimális trajektóriák meghatározhatók a min cut-max flow algoritmus segítségével. Ennek az eljárásnak a lényege, hogy úgy próbáljuk n külön részre vágni a gráfot, hogy az elvágott élek súlyai minimálisak legyenek, közben viszont mind az n részgráfban legyen összeköttetés a kezdeti és a végső csomópont között, és ezeknek az éleknek a súlya pedig legyen maximális.



4.31. ábra. *A több képkockán történő követés gráfja. A gráf minden szintje egy képkockához tartozik, minden szinten egy csomópont pedig egy objektum észlelés.*

További Olvasnivaló

[1] R. Szeliski, *Computer Vision*. Springer London, 2011. DOI: 10.1007/978-1-84882-935-0. cím: https://doi.org/10.1007%2F978-1-84882-935-0.

[2] L. Ross, "The Image Processing Handbook, Sixth Edition, John C. Russ. CRC Press, Boca Raton FL, 2011, 972 pages. ISBN 1-4398-4045-0(Hardcover)", *Microscopy and Microanalysis*, 17. évf., 5. sz., 843–843. old., 2011. szept. DOI: 10.1017/s1431927611012050. cím: https://doi.org/10.1017%2Fs1431927611012050.

[5] J. Shi és Tomasi, "Good features to track", *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition CVPR-94*, IEEE Comput. Soc. Press, 1994. DOI: 10.1109/cvpr.1994.323794. cím: https://doi.org/10.1109%2Fcvpr.1994.323794.

[6] C. Harris és M. Stephens, "A Combined Corner and Edge Detector", *Procedings of the Alvey Vision Conference 1988*, Alvey Vision Club, 1988. DOI: 10.5244/c.2.23. cím: https://doi.org/10.5244%2Fc.2.23.

[7] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints", *International Journal of Computer Vision*, 60. évf., 2. sz., 91–110. old., 2004. nov. DOI: 10.1023/b:visi.0000029664.99615.94. cím: https://doi.org/10.1023%2Fb%3Avisi.0000029664.99615.94.

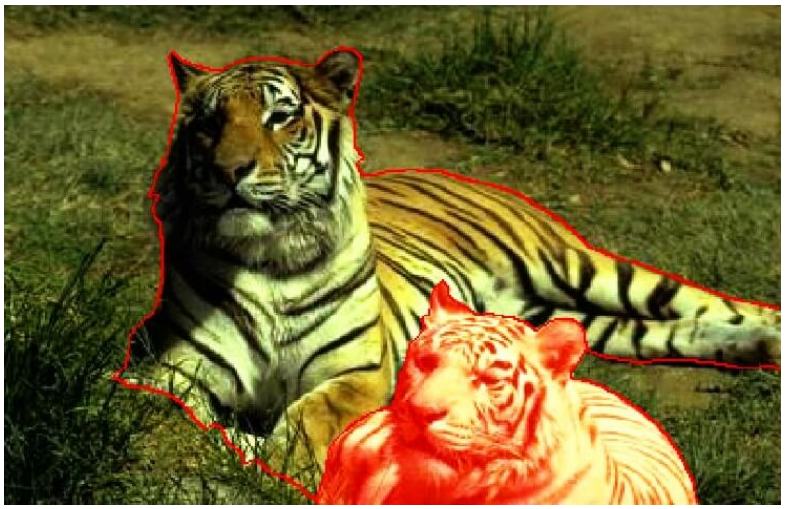
[8] E. Rublee, V. Rabaud, K. Konolige és G. Bradski, "ORB: An efficient alternative to SIFT or SURF", *2011 International Conference on Computer Vision*, IEEE, 2011. nov. DOI: 10.1109/iccv.2011.6126544. cím: https://doi.org/10.1109%2Ficcv.2011.6126544.

5. fejezet

Szegmentáció

5.1. Bevezetés, módszerek

A számítógépes látás megoldásai során sok esetben van arra szükségünk, hogy a kép pixeleit egyesével különálló objektumokhoz soroljuk, vagyis a képet szegmentáljuk. A szegmentálás kimenete leggyakrabban egy két- vagy többállapotú kép, amely adott esetben akár maszkként is felhasználható arra, hogy az eredeti képen található objektumot a képből kiemeljük vagy kivágjuk. A szegmentálás művelete felhasználható az előző alfejezet végén említett objektumjavasló módszerek létrehozására is.



5.1. ábra. *A szegmentálás problémája.*

A szegmentáláshoz felhasznált információk és az eljárás konkrét kimenete alapján a szegmentálás többféle változatát különböztethetjük meg. Előtér-szegmentálásról beszélhetünk, ha valamilyen ismert tulajdonságú objektumhoz tartozó pixeleket kívánunk megkeresni. Hagyományos képszegmentálás esetén viszont nincs kitüntetett előtérobjektum, hanem a képet egyszerűen különálló objektumok szerint kívánjuk szétválasztani. Előfordul, hogy a képet nem különálló objektumok, hanem objektumkategóriák, -osztályok szerint szegmentáljuk, ekkor szemantikus szegmentálásról beszélünk. Ez a mód azonban ugyanazon osztály két szomszédos objektumát egyetlen szegmenssé olvasztja össze. Amennyiben nemcsak osztályok, hanem azon belül külön példányok szerint is szeretnénk szegmentálni, akkor példányszegmentálásról beszélünk.

A szegmentálásnak számos módszere létezik, melyek közül a legegyszerűbbek a szín, vagy intenzitás alapú módszerek, például a küszöbözés és a klaszterezés. Léteznek azonban olyan módszerek,

amelyek különböző egybetartozási kritériumok alapján igyekeznek összefüggő régiókat keresni, ezeket régió alapú módszereknek nevezzük. Léteznek ezen felül még él, mozgás és gráfvágás alapú módszerek is.

5.2. Küszöbözés

Előtér-szegmentálást legegyszerűbb szín vagy intenzitás alapján végezni. Egyszerű, kontrollált környezetekben gyakran garantálható, hogy az alkalmazásunk számára releváns objektum lesz az egyetlen meghatározott színű tárgy a képen, így egyszerűen egy, a színcsatornákon elvégzett küszöbözés segítségével a szegmentálás elvégezhető. Természetesen a valóságban az így kapott bináris képet még a 4.4. alfejezetben ismertetett módszerek segítségével szűrni kell. Fontos megjegyezni, hogy akár intenzitás-, akár színalapú szegmentálásról beszélünk, a küszöbözést szinte kizárólag a HSV- vagy a YCbCr-színterek valamelyikében (vagy ezek variációiban) végzik el.

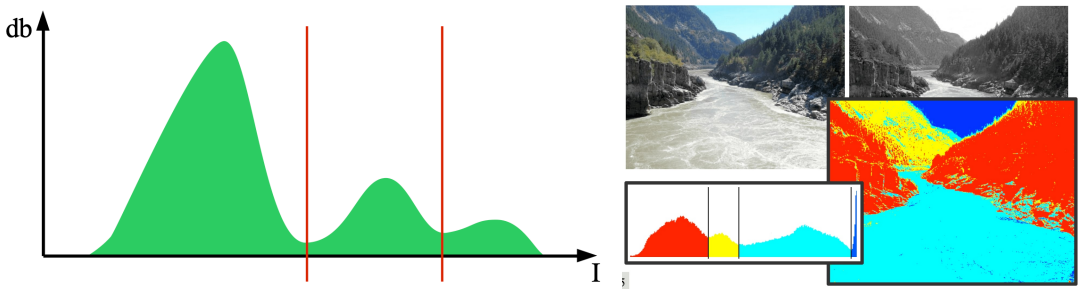
5.2. ábra. *A küszöbözés (bal) és a többértékű küszöbözés (jobb) eredménye.*

A küszöbözés alapú szegmentáció esetén gyakran a tervezők határozzák meg a használt küszöbértéket, amely gyakran szuboptimális lehet. Ennek elkerülésére gyakorta alkalmazzuk a hisztogram-visszavetítés módszerét. Az eljárás elején a szegmentálandó objektumról egy referenciahisztogramot készítünk, amelyet aztán a működés során a képpel összehasonlítunk. A képen a visszavetítés során minden pixelhez meghatározzuk, hogy mekkora eséllyel származik a referenciahisztogramból. Az így kapott valószínűségi képet küszöbözve kaphatunk egy maszkot, amely az objektumhoz tartozó pixeleket tartalmazza.

A szegmentálandó kép hisztogramját felhasználhatjuk hagyományos szegmentáláshoz is, referenciaobjektum nélkül. Ekkor az algoritmus a kép hisztogramjában völgyekkel elválasztott csúcsokat keres, ily módon több részre osztva a kép pixeleit. Ezt a szegmentálást természetesen több dimenzióban egyszerre is el lehet végezni, amely esetben mind a szín-, mind a világosságinformációt felhasználhatjuk.

Alkalmazás: Számos virtuális- és kiterjesztettvalóság-rendszer használ kézi gesztusokat az ember–gép kommunikáció egyik irányának megvalósítására. Bár léteznek olyan megoldások, ahol különféle érzékelőkkel ellátott kesztyűk segítségével érzékeljük ezeket a gesztusokat, legalább ugyanannyi kameraalapú gesztusfelismerő rendszer létezik. Ezek működéséhez azonban szükséges a kéz szegmentálása, amelyet könnyű például a hisztogram-visszavetítéses módszer alapján, a bőr színének segítségével elvégezni. Fontos megjegyezni, hogy ezt a megoldást csak akkor célszerű alkalmazni, ha a kézen kívül a kamera nem lát más bőrfelületet, mivel ebben az esetben azt is kéznek tekinthetjük. Ez általában garantálható fejre szerelt látórendszerek (például kamerával rendelkező megjelenítő szemüvegek) esetében.

A szín alapú szegmentáció egyik nagy problémája az, hogy a színek értelmezése bizonyos esetekben szubjektív lehet. Ezt jól illusztrálja a 2015-ben az internetet meghódító Dress (ruha) nevű kép. Ez



5.3. ábra. *A hisztogram alapú küszöbözés elve (bal) és eredménye (jobb).*

egy olyan két színből álló ruhát ábrázoló kép, amit az emberek egy része fehér és arany, míg a másik részük kék és fekete színűnek lát. A jelenség pontos magyarázata máig ismeretlen, a legnépszerűbb elmélet azonban a természetes fény színtorzító hatásának egyedenként eltérő kompenzációjára vezeti vissza.

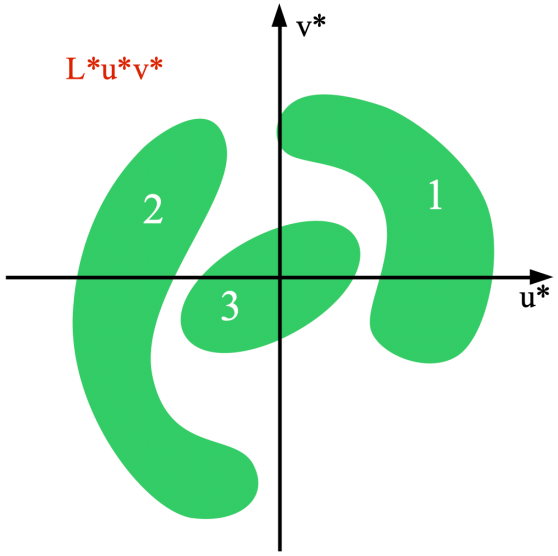
5.4. ábra. *A Dress.*

5.3. Klaszterezés

A fenti, hisztogramalapú szegmentáláshoz rendkívül hasonló megoldás a klaszterezésre épülő szegmentálás. Korábban, a vizuális szóhalmazos osztályozás során már említettük a klaszterezést, amelynek lényege, hogy egy tetszőleges térben értelmezett ponthalmaz pontjait úgy osztjuk be valahány részhalmazba, vagyis klaszterbe, hogy az így kapott klaszterek valamilyen kompaktsági kritériumot minél inkább kielégítsenek. Bár klaszterezésre számos algoritmust javasoltak, az egyik legelterjedtebb megoldás a k-közép-eljárás (angolul: k-Means).

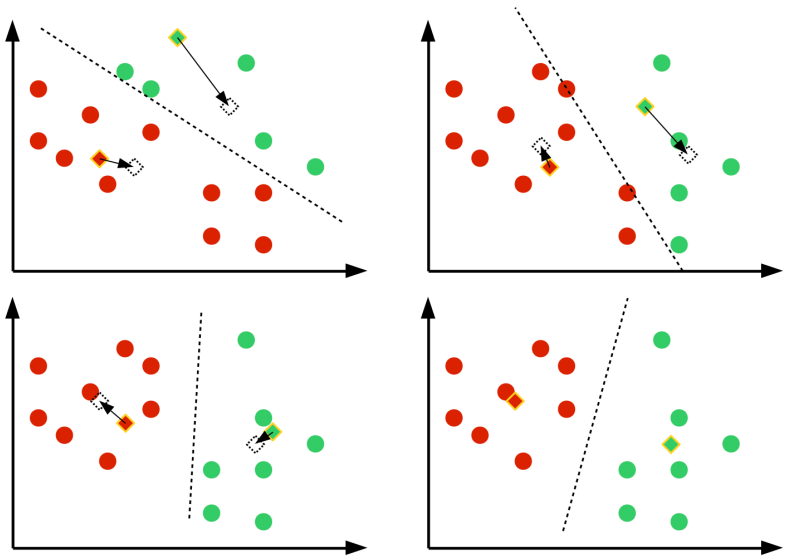
5.3.1. K-means

A k-közép-módszer a ponthalmazt úgy kívánja k darab klaszterbe sorolni, hogy e klaszterek elemeinek a klaszter középpontjától számított négyzetes távolságainak összege minimális legyen. Ehhez egy iteratív algoritmust használ, amelynek inicializáló lépése, hogy az adatpontok által kifeszített térbe véletlenszerűen lerak k darab középpontot. Ezt követően az algoritmus konvergenciáig ismételi a következő két lépést. Első lépésként minden pontot hozzárendel a hozzá legközelebb lévő klaszterközépponthoz, így az egyes pontok klaszter-hozzárendelését megváltoztatja. Ezt követően



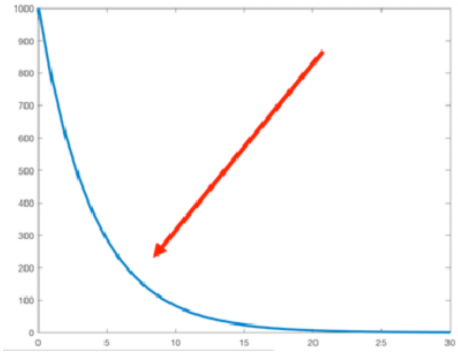
5.5. ábra. *A klaszterezés elve.*

az egyes klaszterközéppontoknak új értéket ad, ami az adott klaszterhez tartozó pontok számtani közepe lesz. Ez a lépés megváltoztatja a középpontok helyzetét, így azt is, hogy melyik pont melyik klaszterhez tartozik, így az iteráció tovább futtatható.



5.6. ábra. *A k-means elve.*

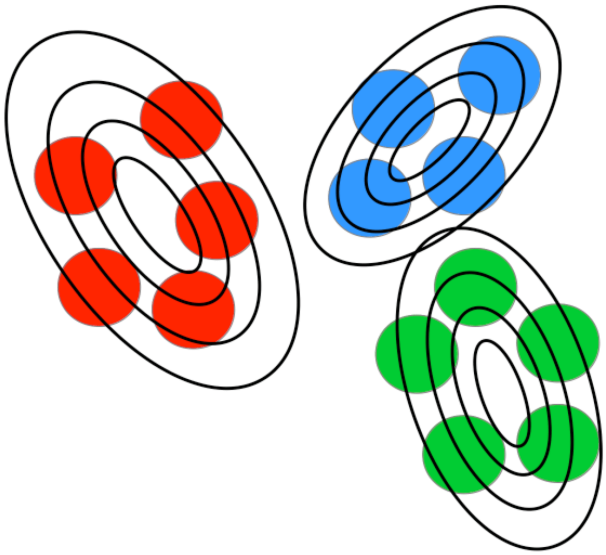
A k-közép-módszer bizonyítottan konvergál a véletlen inicializálás konkrét értékeitől függetlenül. Leállási feltételnek választható a klaszterközéppontok vagy a ponthozzárendelések állandósága (vagy adott esetben, ha a változás egy küszöbérték alatt van). Fontos megjegyezni, hogy a módszer nevében és működésében szereplő k egy, a tervező által választott érték, amelynek körültekintő megválasztása nagymértékben befolyásolja az algoritmus működését. A probléma ugyanis az, hogy a k növelésével minden esetben lehetséges a kapott klaszterek kompaktságát növelni, és valóban, N darab pont beosztható N darab klaszterbe zérus hiba mellett. Ez persze azt jelentené, hogy egy kép összes pixelét külön objektumhoz soroljuk, amelynek csekély értelme van. Érdemes ezért megjeleníteni a hiba csökkenését a klaszterek számának függvényében, és kiválasztani ennek a függvénynek a könyökpontját, vagyis azt a klaszterszámot, amit növelve a hiba már nem csökken tovább számottevően.



5.7. ábra. *A klaszterek számának megválasztása. Bár a klaszterezés hibája monoton csökken, érdemes az ábrán jelölt könyökpontot választani, hiszen eddig érünk el jelentős javulást a klaszterszám növelésével.*

5.3.2. Mixture of Gaussians

A k-Means módszer során minden egyes pontot pontosan egy klaszterhez rendelünk hozzá. Ez azonban nem feltétlen kell, hogy így legyen, egyes pontok tartozhatnak több klaszterhez is eltérő valószínűségekkel. Ebben az esetben soft, vagy fuzzy klaszterezésről beszélünk. Ilyen algoritmus például a Mixture of Gaussians (MoG), amely a ponthalmazt k db független gauss eloszlás segítségével írja le. A módszer célja, hogy meghatározza az eloszlások paramétereit úgy, hogy a ponthalmaz valószínűsége maximális legyen. Ehhez az úgynevezett Expectation-Maximization (EM) algoritmust használja.



5.8. ábra. *A MoG klaszterezés elve. Az ábrán az egyes klasztereket leíró gauss eloszlások szintvonalait láthatjuk.*

Az EM egy két lépéses iteratív algoritmus, amely a keresendő normális eloszlásokat véletlen módon inicializálja, majd a konvergenciáig az alábbi két lépést ismétli:

1. **Expectation:** Minden pontot hozzárendelünk ahhoz a normális eloszláshoz, amelyik a legnagyobb valószínűséget adja az adott pontnak.
2. **Maximization:** Az egyes eloszlások paramétereit megbecsüljük a maximum-likelihood módszer segítségével a hozzárendelt pontokból.

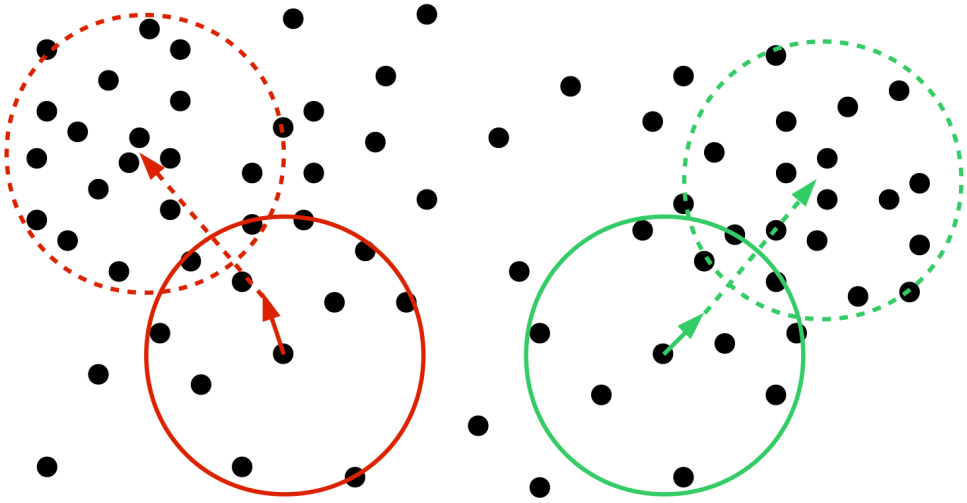
Az X pontokat tartalmazó normális eloszlás paraméterei az alábbi módon becsülhetők (Maximum Likelihood):

$$\begin{aligned} \hat{\mu} &= \overline{X} \\ \hat{\Sigma} &= \frac{1}{N-1}(X-\hat{\mu})^T(X-\hat{\mu}) \end{aligned} \tag{5.1}$$

Látható, hogy az EM és a k-means lépései meglehetősen hasonlóak. Lényeges különbség azonban, hogy a MoG eredményeképp eltérő szélességű és formájú klaszterek adódhatnak, míg ez a k-Means során nem lehetséges.

5.3.3. Mean Shift

Az utolsó fontos klaszterező eljárás a mean-shift, melynek során először meghatározunk egy súlyozó kernelt. Képszegmentálás során ez leggyakrabban egy sima (Flat), vagy egy Gauss kernel szokott lenni. Szükséges még a kernel méretét is meghatározni, amely befolyásolja a végső szegmentáció finomságát. Fontos megjegyezni azonban, hogy a mean-shift során nem szükséges meghatározni a klaszterek számát, ez ugyanis automatikusan kialakul az algoritmus futása során. Ez jelentős előnyt jelent az előző két módszerrel szemben.

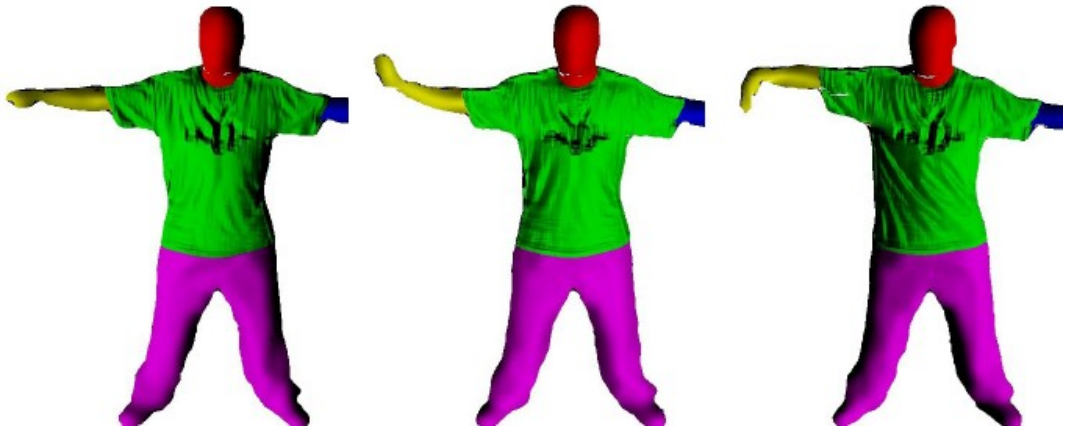


5.9. ábra. *A mean-shift klaszterezés elve.*

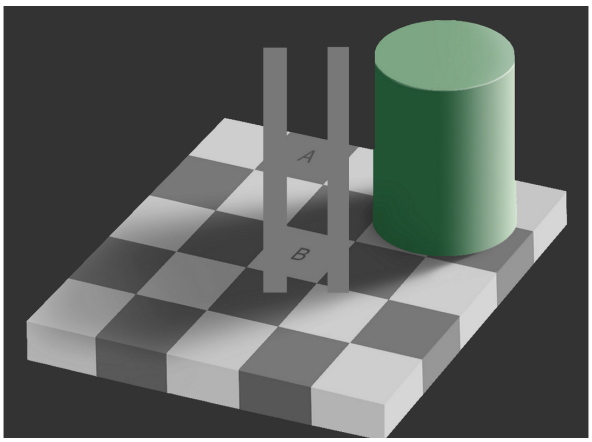
A mean-shift futása során a kernellel végighaladunk a pontokon, majd minden pozícióban kiszámoljuk a kernel által lefedett pontok átlagát (a kernel súlyainak megfelelően). Ezt követően az adott pontot az így kiszámított átlagba toljuk el. Ezt a műveletet addig ismételjük, amíg van változás.

5.4. Régió alapú módszerek

Az eddig felsorolt módszereknek az egyik legnagyobb hátránya, hogy csupán az intenzitás- vagy színinformációt veszik figyelembe. Mivel az univerzum legtöbb objektuma térben összefüggő (ebből következően a képük is összefüggő lesz), ezért alapvető kritérium, hogy a szegmentálás során kapott szegmensek is ilyenek legyenek. Erre a problémára adnak megoldást a különböző régió-alapú módszerek, például a régiónövesztés és a régiószeletelés módszere, valamint a gráfvágásokon alapuló szegmentálási módszerek.



5.10. ábra. *A mean-shift algoritmus eredménye.*



5.11. ábra. *Az intenzitás/szín használatának hátránya: A különböző valós hatások miatt a képen nyilvánvalóan különböző régiók azonos intenzitással rendelkeznek, ami azonos klaszterbe tartozást eredményezne.*

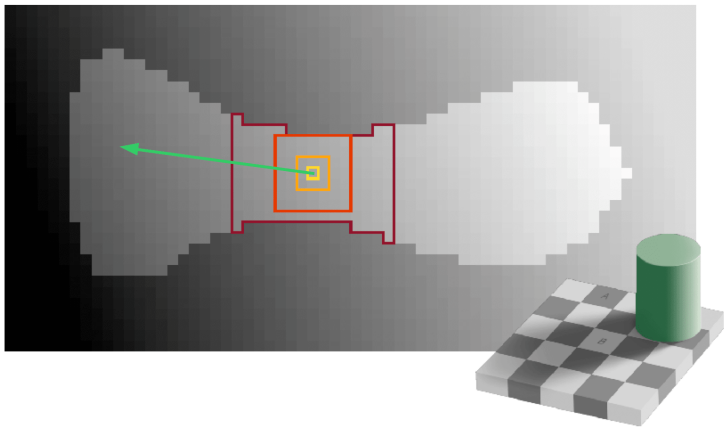
5.4.1. Régiónövelés

A régiónövesztés eljárása esetén az algoritmus első lépésként kiválaszt a képről néhány magpontot. Ezeknek a régiómagoknak a választása történhet a képpontok intenzitása alapján vagy például egy rácshálón egyenletesen elhelyezkedve. A régiómagok lesznek az egyes régiók kezdeti értékei. Ezekből a magokból kiindulva megvizsgáljuk minden régió még címkézetlen szomszédjait, és kiértékelünk egy régiótagsági függvényt. Amennyiben ez a tagsági függvény pozitív eredményt ad vissza, akkor a vizsgált pontot hozzáadjuk a régióhoz, ellenkező esetben nem. Az algoritmus leáll, ha már egyetlen pontot sem tudunk egyik régióhoz sem hozzáadni.

A régiónövesztéses algoritmus eredményét számos tényező befolyásolja. Ezek közül az egyik a kiindulópontok megválasztása, amelyeket célszerű a kép hisztogramjának felhasználásával kiválasztani. Érdemes továbbá az eredményben kapott, esetlegesen túl kicsi régiókat elvetni, mert ezek nagy valószínűséggel valamilyen lokális zaj vagy hiba termékei. A legfontosabb tényező azonban a régióhoz tartozás kritériuma. Leggyakrabban ezt a szomszédos pixelek közötti intenzitáskülönbség vagy a régi középpont és a vizsgált pixel közötti intenzitáskülönbség alapján döntjük el, egy küszöbérték alapján. Speciális képek esetén fel lehet még használni a szín vagy textúrabeli hasonlóságokat.

5.4.2. Split & Merge

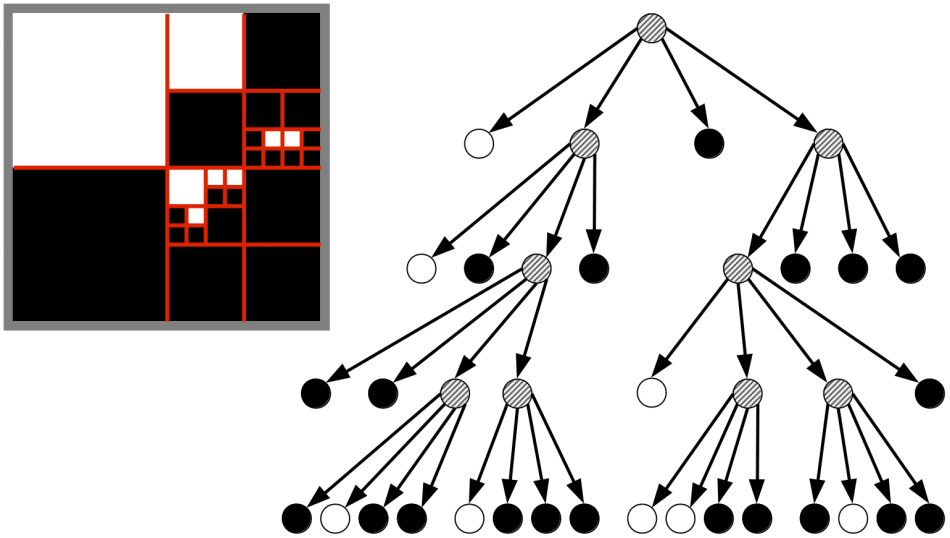
Ezekre a problémákra talál megoldást a régiószeleteléses eljárás, amely kiindulási állapotában az egész képet egyetlen összefüggő szegmensnek tekinti. Ezt követően iteratív módon végighalad az



5.12. ábra. *A régiónövesztés elve.*

összes szegmensen, és ha a szegmens megfelel egy, a tervező által megadott homogenitási kritériumnak, akkor érintetlenül hagyja. Ellenkező esetben a szegmenst négy, egyenlő területű részre osztja, majd ezeket a szegmenseket is megvizsgálja. Ezt addig folytatja, amíg még keletkeznek új szegmensek.

A szeletelés után viszont jó eséllyel szétválasztunk számos, egyébként egybetartozó régiót is. Éppen ezért a szeletelés befejezése után egy összeolvasztási fázis is következik, amely szintén egy hasonlósági kritérium alapján összevon szomszédos régiókat, amennyiben ez szükséges. E módszer használatával sikerült egy gyors, globális információt használó eljárást alkotni, amely lényegesen robusztusabb a zajokra, és majdnem teljesen invariáns a szomszédsági választásra. A módszer hátránya, hogy a szegmensek határa nem mindig lesz teljesen pontos a négyzetes módon végzett szeletelés miatt.



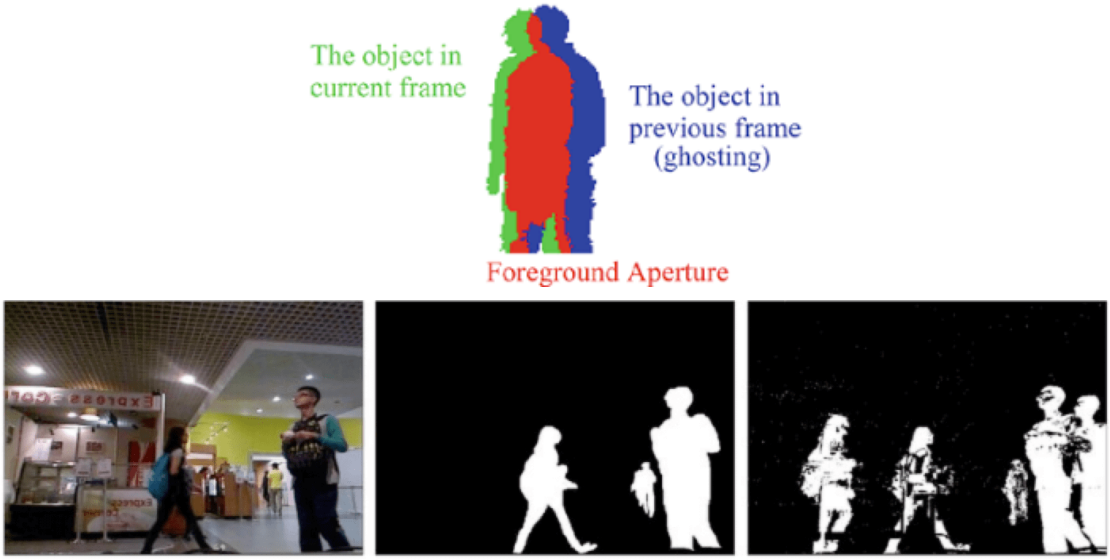
5.13. ábra. *A Split & Merge algoritmus elve.*

5.5. Mozgás szegmentáció

Videók esetében fontos feladat a mozgás észlelése és szegmentálása, amely tulajdonképpen az előadás elején bemutatott előtér-szegmentáció egyik formájának is tekinthető, ahol a szegmentálás alapja nem a szín vagy az intenzitás, hanem a mozgás. A mozgás detektálásának egyik legegyszerűbb módja az eltérő időpillanatban (általában nem rögtön egymás utáni, hanem 0.5–1 másodperc

különbséggel) készült különbségkép képzése, amellyel ki lehet emelni azokat a pixeleket, amelyek a képkockák között jelentős változáson mentek keresztül. Az így kapott különbségképet küszöbözzük, így a kapott bináris mozgásképen az „1” pixelek területét kiszámítva dönthetünk arról, hogy érzékeltünk-e mozgást a képen.

A fent leírt, rendkívül egyszerű módszernek számos hátránya van, amelyek közül az egyik, hogy egy mozgó objektum esetén a módszer az objektum mindkét képen lévő pozíciója helyén mozgást fog érzékelni, így a kapott szegmentálásra is használandó mozgásképen „szellemképes” lesz, vagyis egy mozgó objektum két helyen is megtalálható lesz. A másik probléma, hogy számos olyan esemény történhet a videón, amelyet nem tekintünk mozgásnak, mégis jelentős különbséget okoz az egymást követő képkockák intenzitásértékeiben. Erre nagyszerű példa a megvilágítás megváltozása, amely a szabad téren készült videók esetében az egyik legfontosabb zavaró tényező. Beltéri videók esetén is okozhat ilyen változást például a kamera automatikus fehéregyensúly-állító funkciója.

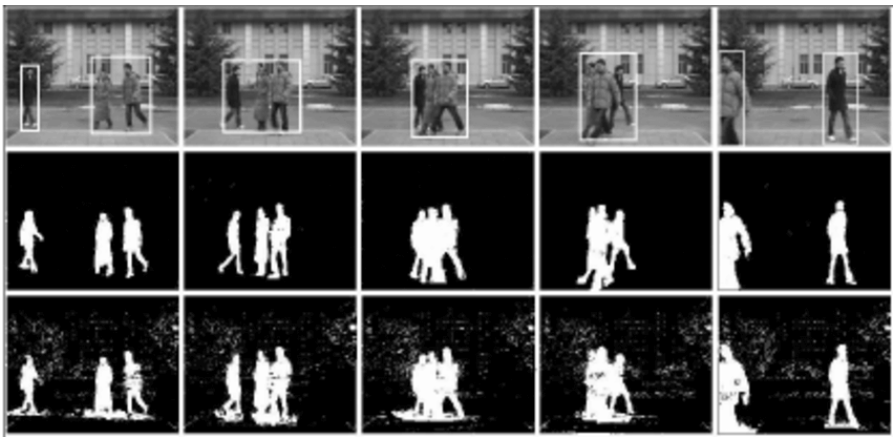


5.14. ábra. *A szellemkép megjelenése.*

A fenti problémák kiküszöbölésére való a Gauss-háttérmodellen alapuló mozgásdetektálás. Ennek az eljárásnak a célja, hogy egy statikusnak tekintett hátteret a mozgó előtértől képes legyen elválasztani. Ehhez a háttérről egy statisztikai modellt készít egy mozgóátlagoló eljárás segítségével, és minden egyes pixelhez kiszámítja a pixel értékének várható értékét és szórását. Ezt követően a működés során a különbségképet az aktuális kép és a háttérkép között végezzük, és ezt küszöbözzük. Az így elért eredmények már nem tartalmaznak szellemképet, valamint a fokozatosan végbemenő intenzitásváltozások (például hogy kisüt a nap) be tudnak épülni a háttérbe hamis mozgás detektálása nélkül.

A háttérmodellben minden pixelhez kiszámolt szórásértéket fel lehet használni az adaptív küszöbérték meghatározására. Ily módon a modell automatikusan lehet érzéketlenebb olyan területeken, ahol gyakori a mozgás, és érzékenyebb ott, ahol ez ritkább. Ez a megoldás olyan esetekben lehet hasznos, ha sok olyan terület látszik a képen, ahol gyakori, irreleváns mozgást érzékelhetünk (például szeles időben a bokrok, illetve fák képe).

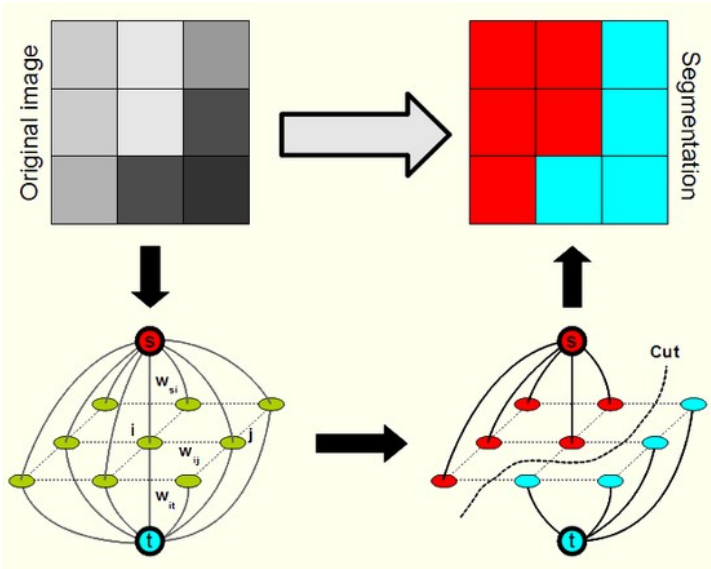
Alkalmazás: Térmegfigyelési és biztonsági alkalmazások esetén fontos az elkészült felvételek tárolása a későbbi felhasználás érdekében. Ez általában rendkívüli méretű tárhelyet igényel, nagy részben a videók mérete és a megkövetelt redundancia miatt. A szükséges tárhely azonban nagymértékben csökkenthető, ha csak azokat a felvételrészleteket tároljuk el, ahol mozgást érzékeltünk, mivel azok a felvételek, ahol semmi sem történt, nyilvánvalóan nem értékesek számunkra. E rendszerek működését nagymértékben javítja a zajokra robusztusabb háttérmodelles eljárás alkalmazása, különösképpen kültéri felvételek esetén.



5.15. ábra. *A gauss háttérmodell alapú mozgásszegmentálás elve. A középső sorban a varianciа alapú különbségküszöbözéssel előállított képek látszanak, míg az alsó sorban az egyszer konstans küszöbbel készített eredmények.*

5.6. Graph Cut

Érdemes megemlíteni még a gráfvágásra alapuló szegmentálási módszereket, amelyeknek számos változata létezik. E módszerek közös tulajdonsága, hogy a képet egy súlyozott, irányítatlan gráfként írják le, ahol a gráf csomópontjai a pixelek vagy a lokális pixelcsoportok. A gráf élei csak a szomszédos pixelek vagy csoportok között vannak megadva, a rajtuk szereplő súlyok pedig a pixelek vagy csoportok közötti különbözőséget fejezik ki. A legtöbb ilyen eljárás lényege, hogy a konstruált gráfot olyan módon vágják több részre, hogy e vágások költsége minimális legyen. A vágások költségét természetesen a gráfból kitörölt éleknek a súlyai alapján határozzák meg.



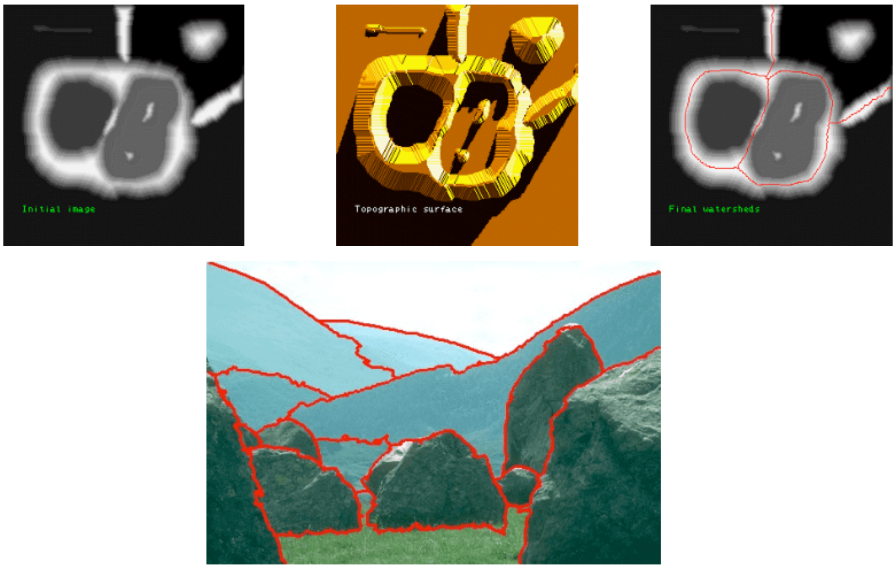
5.16. ábra. *A gráfvágásos szegmentálás elve.*

Alkalmazás: Érdemes megjegyezni, hogy szegmentálási módszereket gyakran alkalmaznak úgynevezett szuperpixel-szegmentációra is. A szuperpixel egy relatíve kisméretű, kompakt képrészlet, amely valamilyen homogenitási tulajdonságokkal rendelkezik. Nagyméretű, összetett objektumok általában számos szuperpixelből állnak. A szegmentáció ilyen módon történő elvégzésére ugyanezek az algoritmusok használhatók, csupán a paramétereiket kell úgy beállítani, hogy sok, kisméretű szegmenst találjanak. Léteznek természetesen kifejezetten erre kifejlesztett eljárások. A szuperpixelek rendkívül hasznosak számos alkalmazásban, például a képek kézzel történő felcímkézését (amely szükséges a gépi tanuláshoz való, tanító adatbázisok előállításához) lehet gyorsítani ezzel a

módszerrel.

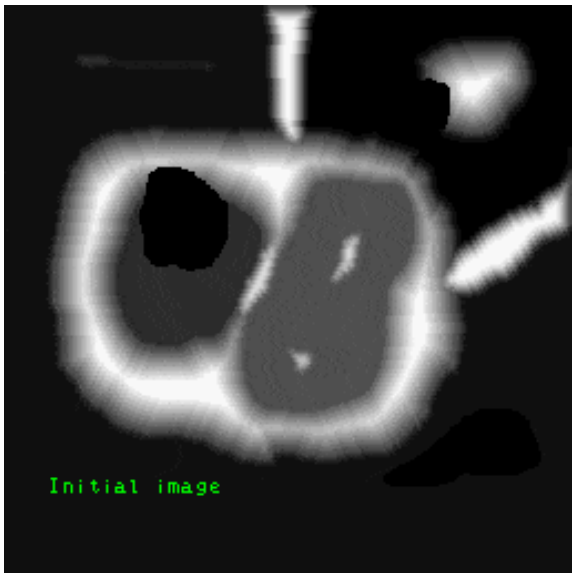
5.7. Watershed

Az utolsó népszerű algoritmus a vízválasztó (Watershed) eljárás. Ennek a módszernek lényege, hogy kép intenzitás értékeit úgy képzeljük el, mint egy szint térképet (vagyis minél világosabb egy pixel, annál magasabban helyezkedik el a képsík felett). Ezt követően a képen megkeressük a lokális minimumokat, majd mindegyikből egyszerre kezdjük el "elárasztani" a képet. Amikor két külön minimumból induló árasztás összeér, akkor megtaláltuk a vízválasztó vonalat, vagyis a két szegmens közti határt.



5.17. ábra. *A Watershed algoritmus elve.*

A Watershed algoritmus egyik nagy előnye, hogy könnyen lehetővé teszi a kézi beavatkozást, ugyanis az árasztások kiinduló pontjait kézzel is meghatározhatjuk. Ilyen módszerrel nehéz képek esetén is meglehetősen jó minőségű szegmentációt lehet elérni.



5.18. ábra. *A markeres watershed algoritmus elve.*

További Olvasnivaló

[1] R. Szeliski, *Computer Vision*. Springer London, 2011. DOI: 10.1007/978-1-84882-935-0. cím: https://doi.org/10.1007%2F978-1-84882-935-0.

[2] L. Ross, "The Image Processing Handbook, Sixth Edition, John C. Russ. CRC Press, Boca Raton FL, 2011, 972 pages. ISBN 1-4398-4045-0(Hardcover)", *Microscopy and Microanalysis*, 17. évf., 5. sz., 843–843. old., 2011. szept. DOI: 10.1017/s1431927611012050. cím: https://doi.org/10.1017%2Fs1431927611012050.

[14] Y. Cheng, "Mean shift, mode seeking, and clustering", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17. évf., 8. sz., 790–799. old., 1995. DOI: 10.1109/34.400568. cím: https://doi.org/10.1109%2F34.400568.

[15] R. Nock és F. Nielsen, "Statistical region merging", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26. évf., 11. sz., 1452–1458. old., 2004. nov. DOI: 10.1109/tpami.2004.110. cím: https://doi.org/10.1109%2Ftpami.2004.110.

[16] C. Couprie, L. Grady, L. Najman és H. Talbot, "Power Watershed: A Unifying Graph-Based Optimization Framework", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33. évf., 7. sz., 1384–1399. old., 2011. júl. DOI: 10.1109/tpami.2010.200. cím: https://doi.org/10.1109%2Ftpami.2010.200.

[17] L. Najman és M. Schmitt, "Watershed of a continuous function", *Signal Processing*, 38. évf., 1. sz., 99–112. old., 1994. júl. DOI: 10.1016/0165-1684(94)90059-0. cím: https://doi.org/10.1016%2F0165-1684%2894%2990059-0.

6. fejezet

Bináris képek

6.1. Bevezetés

Már említettük korábban, hogy a képeknek több fajtája létezik. Ezek közül a leggyakoribbak az egycsatornás szürkeárnyalatos, illetve a háromcsatornás színes képek. Szintén gyakran előfordulnak azonban kétállapotú, bináris képek, például éldetektálás esetén, ahol az élekhez tartozó pixelek egyes értéket vettek fel, míg a többi nullást. Ilyen bináris képek azonban számos más művelet eredményeképp is előállhatnak, például küszöbözés, színdetektálás vagy komplex objektumdetektáló eljárások során.

A jelenlegi alfejezetben olyan bináris képeket fogunk vizsgálni, amelyeknek a két állapota közül az egyik az „1”, ami az adott alkalmazás szempontjából relevánsnak tekintett objektumainkat jelöli, míg a „0” a számunkra irreleváns hátteret reprezentálja. Fontos megjegyezni, hogy a megjelenítés során a láthatóság kedvéért a bináris képek két állapotát a maximális fehér és a minimális fekete színek szokták jelölni, arra viszont nincs egyértelmű konvenció, hogy a két szín közül melyik jelöli az objektumot, és melyik a hátteret. A jelen műben következetesen a fehér szín fogja az objektumot jelölni, de ez más forrásokban lehet fordítva is.

6.2. Morfológia

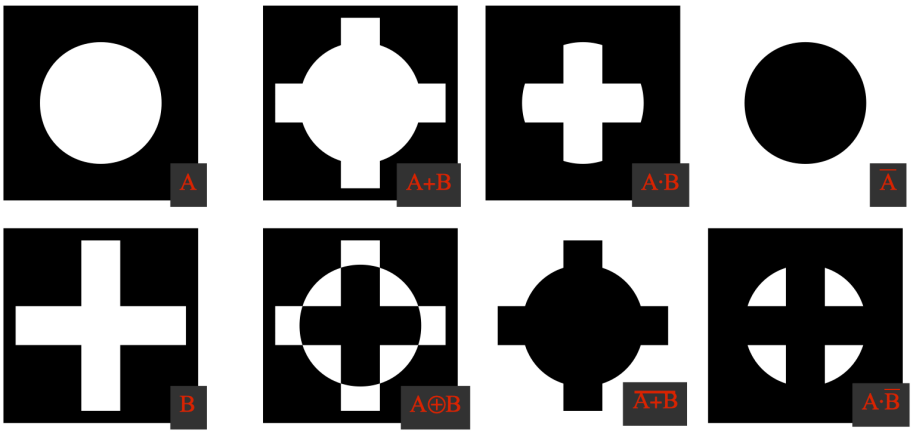
Az első fontos algoritmuscsalád a bináris morfológia, vagyis alakzattan csoportjába tartozik. Ezek az algoritmusok elsősorban javítások elvégzésére használhatók bináris képeken.

6.2.1. Erózió és dilatáció

A gyakorlatban bármilyen kifinomult módszert is használunk az objektumok elkülönítésére, ez nem fog tökéletesen sikerülni, így – ahogy a színes és szürkeárnyalatos képek esetében is – szükség van a bináris képek javítására. Természetesen mivel a bináris képek hibái más jellegűek, ezért az azok javítására használt módszerek is jelentősen eltérnek. A bináris képeknek alapvetően két jellemző hibája fordul elő: az egyik az olyan pixelek jelenléte, amelyek a valóságban a háttérhez tartoznak, azonban mégis objektumként lettek címkézve, valamint ennek az ellentéte: a tévesen háttérként címkézett objektumpixelek. Az előbbi hibák miatt hamis objektumokat láthatunk a képen, vagy a valóságban elkülönülő objektumokat tévesen összenövesztethetünk. Az utóbbi miatt előfordulhat, hogy lukakat kapunk egyes objektumokon belül, vagy egy a valóságban egybefüggő objektumot tévesen szétválasztunk.

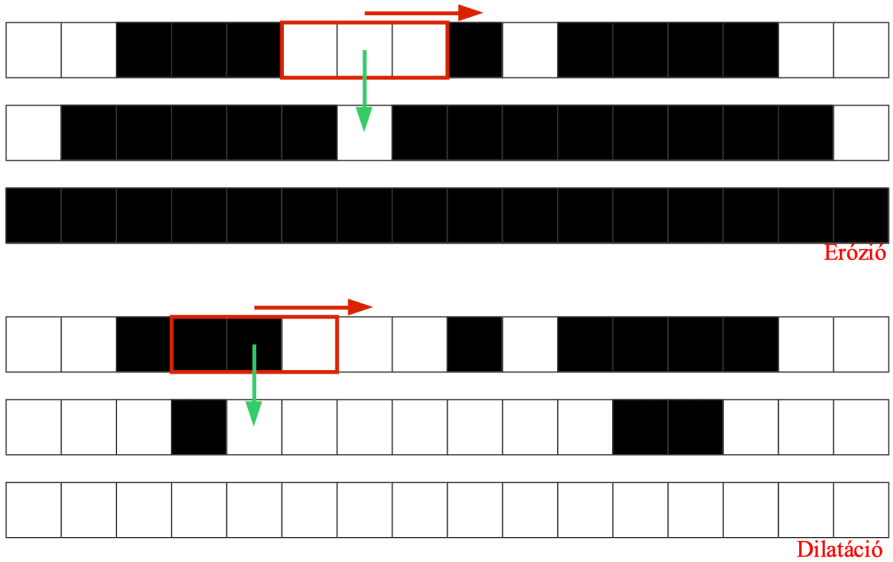
A fenti két bináris képi hibára használható az erózió és a dilatáció művelete. Mindkét művelet elvégzéséhez egy strukturáló elem definiálására van szükségünk. A strukturáló elem egy tetszőleges

alakú ablak, amelyet a konvolúció műveletéhez hasonló módon minden egyes pozícióban ráilleszt-tünk a képre, és valamilyen logikai műveletet végzünk el a segítségével. A konvolúciós kernelekkel szemben azonban a strukturáló elem értékei csak „0”, „1”, illetve esetenként meghatározatlan („Don’t care”) értékek lehetnek. Bináris képek, vagy bináris kép és strukturáló elem között a hagyományos logikai műveleteket definiálhatjuk:



6.1. ábra. *A bináris képek közti logikai műveletek. Ezek a műveletek leggyakrabban pixeleként hajtandók végre.*

Az erózió művelete esetében az eredménykép adott pixelébe akkor írunk „1” értéket, amennyiben a strukturáló elem tökéletesen ráillik a bemeneti képre az adott pozícióban, vagyis minden pixelük értéke megegyezik. Ezzel szemben a dilatáció műveleténél a kimenet akkor lesz „1”, ha a strukturáló elem és a kép legalább egy pozícióban megegyeznek. Amennyiben az erózió műveletét egy csupa „1” strukturáló elemmel végezzük, akkor a művelet során az objektumok szélei nullába állítódnak, így az objektumok mérete csökkenni fog. A dilatáció elvégzése esetén ennek a fordítottja történik, vagyis az egyes objektumok nőni fognak.



6.2. ábra. *Az erózió és dilatáció elve.*

Fontos megjegyezni, hogy ezeket a műveleteket tetszőleges strukturáló elemmel is el lehet végezni, mely esetben különböző speciális szűréseket tudunk végezni a bináris képen. Egy keskeny, téglalap alakú strukturáló elem segítségével például eltüntethetők a bináris képről olyan élszerű objektumok, amelyek iránya a strukturáló elem hosszabbik oldalára merőleges, azonban az elemmel párhuzamos élek megmaradnak. Hasonló módon egy speciális formájú strukturáló elem segítségével a képről kiszűrhető az összes olyan objektum, amelyek formája nem egyezik meg a strukturáló elemével. Ez

a módszer felhasználható például sarkok kiemelésére a bináris képen.

6.3. ábra. *Az erózió és dilatáció művelete végrehajtható szürkeárnyalatos képeken is. Ekkor dilatáció maximum szűrőként (közép) az erózió pedig minimum szűrőként (jobb) viselkedik.*

6.2.2. Nyitás és zárás

Az erózió és dilatáció műveletének nagy hátránya, hogy az objektumok méretét és alakját csökkentik/növelik, így a műveletek alkalmazása után végzett méréseink nem lesznek pontosak. Épp ezért a gyakorlatban sosem szoktuk ezeket az eljárásokat önmagukban alkalmazni, hanem ezeknek kombinációit használjuk. A két leggyakrabban használt eljárás a nyitás és a zárás művelete. A nyitás művelete során először előre meghatározott számú eróziót végzünk el, amely eltünteti a kisméretű, zajszerű objektumokat, majd ezt követően ugyanannyi számú dilatációt csinálunk, amely visszanöveszti a megmaradt objektumokat az eredeti méretükre. Fontos megjegyezni, hogy a nyitásban használt dilatáció olvasztásmentes, vagyis csak akkor állítunk „1” értékbe egy korábban „0” értékű képpontot, ha az nem változtatja meg a független komponensek számát. Így azt is el tudjuk érni, hogy a nyitás a kismértékben összenőtt objektumokat szétválassza.

6.4. ábra. *A kezdeti (fehér) kör alakú objektum számos négyzetes strukturáló elem segítségével elvégzett erózió/dilatáció után jelentős méret- és formabeli változást szenved.*

A zárás művelete a nyitással ellentétes. Először egy meghatározott számú dilatációt végzünk, amelynek segítségével az objektumokon belül keletkező lukakat betömjük, majd ezt követően ugyanannyi erózió segítségével visszaállítjuk az objektumok eredeti méretét. A két műveletet természetesen egymás után is lehet végezni, így mind a két típusú képhibát javítani tudjuk.

6.3. Topológia

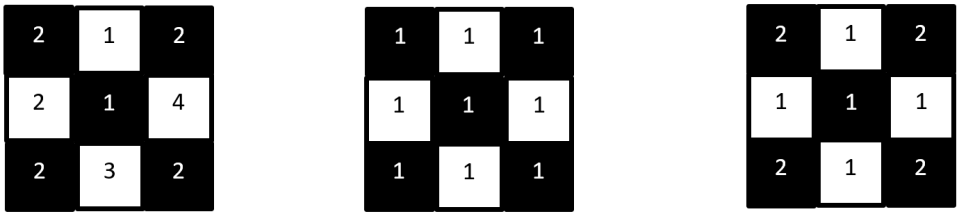
A bináris kép javítása után immáron rendelkezésünkre áll egy meglehetősen jó minőségű kép a számunkra releváns objektumokról, amelyeken különböző méréseket végezhetünk. Ehhez nyújt számunkra nagy segítséget a topológia tudományterülete, amely alapvetően objektumok közti kapcsolatokkal foglalkozik.

6.3.1. Szomszédosság

Mielőtt azonban ezt megtehetnénk, érdemes szót ejteni a pixelek szomszédosságának a kérdéséről. Annak eldöntésére, hogy két pixel szomszédos-e, alapvetően két egyszerű konvenció létezik: a 4, illetve a 8 szomszédosság. A 4 szomszédossági konvenció használata esetén minden pixelnek négy szomszédja van, ezek két oldalt mellette, illetve alatta és fölötte helyezkednek el. A 8 szomszédosság használata esetén az előbbi négy szomszédon felül minden képpontnak további négy szomszédja van, amelyek tőle átlósan helyezkednek el.

Bármelyik szomszédosságkonvenciót is használjuk, az sérteni fogja a kép síkjának az úgynevezett Jordan-tulajdonságát. A Jordan-tulajdonság azt mondja ki, hogy egy síkot egy összefüggő, zárt görbe pontosan két részre oszt szét. Ez a tulajdonság az ember számára magától értetődő, és célszerű lenne, ha a kép síkjában is teljesülne. Azonban az alábbi ábrák közül az elsőn látható, hogy 4 szomszédság használata esetén az adott konfigurációban a fehér színű pixelek egymással nem szomszédosak, mégis a feketével jelölt háttérterületet két részre osztották. A középső ábrán láthatjuk, hogy 8 szomszédosság használata esetén a fehér pixelek egyetlen összefüggő, zárt görbét alkotnak, azonban a háttér pixelei is egy összefüggő részben maradnak.

Ahhoz, hogy a sík Jordan-tulajdonságát megtarthassuk az objektum és a háttér közül az egyikhez 4, a másikhoz pedig 8 szomszédságot kell használnunk. A jobb oldali ábra esetében a fehér előtér objektumhoz 8-szomszédságot használunk, így ezek a pixelek egy zárt görbét alkotnak, a háttérhez azonban 4-szomszédságot, így a középen lévő pixel nem lesz szomszédos egyik fekete pixellel sem, így a háttér valóban két részre oszlik. Az ábrák esetében feltételezzük, hogy az ablak határain túl a háttér folytatódik a végtelenségig, így a sarokokban található fekete pixelek mindig ugyanahhoz az összefüggő részhez tartoznak.

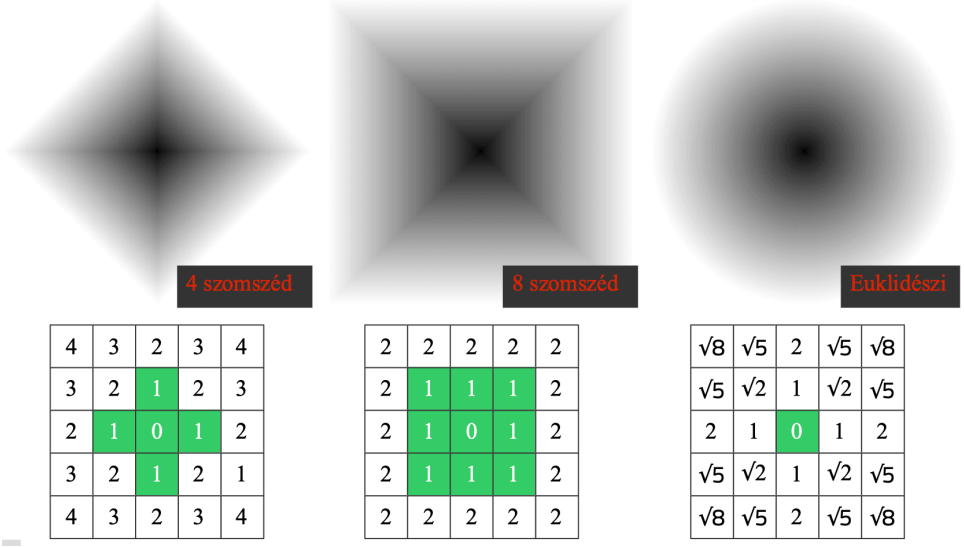


6.5. ábra. *A Jordan-tulajdonság ábrázolása. A bal oldali esetben 4 szomszédosság mellett 2 háttér és 4 előtér komponensünk van. A középső esetben 8 szomszédosságot használtunk, ekkor azonban az előtér és a háttér is összefüggő. A jobb oldali esetben az előtér 8, a háttér pedig 4 szomszédságú, így egy összefüggő előtér és két háttér régiónk van.*

Érdemes megjegyezni, hogy a szomszédossághoz hasonlóan a képtér távolság mércéjét is meg kell határoznunk ahhoz, hogy méréseket (különösen hossz) tudjunk végezni. Erre több lehetőségünk van: választhatunk négyes távolságot, nyolcas távolságot, vagy euklideszi távolságot. Bár az utóbbi lényegesen számításigényesebb, az előző kettő viszont rendkívül nagy pontatlanságokkal járhat.

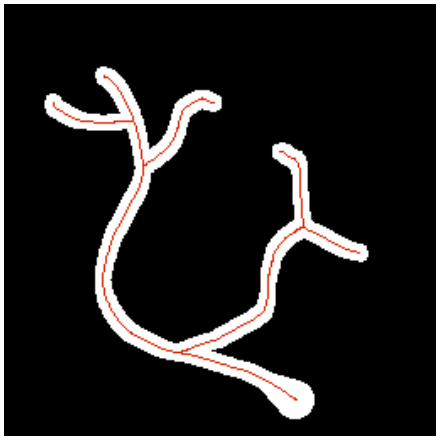
6.3.2. Csontvázasítás

Ennek tisztázása után az objektumok számos tulajdonságát meghatározhatjuk a bináris képen. Ezek közül az első az objektum csontváza. A csontváz alapvetően egy olyan reprezentáció, amelyben



6.6. ábra. *Az egyes szomszédosságokra alapuló és az euklideszi távolság mércék.*

minden ”1” értékű pontnak pontosan 1, vagy 2 ”1” értékű szomszédja van, topológiailag azonban megegyezik az eredeti objektummal. Alternatív módon a csontváz elképzelhető úgy is, mint az objektumba helyezhető maximális átmérőjű körök origóinak összessége.



6.7. ábra. *Egy bináris objektum csontváza.*

Az objektum csontvázát leggyakrabban valamilyen iteratív eróziós eljárással határozzák meg (vékonyítás). Ebben az esetben azonban az eróziót csak akkor végezzük el, ha nem törlünk vele lényeges pontokat, vagyis olyanokat, amelyek végpontok, vagy a törlésük az objektumot kettészakítaná. Fontos megjegyezni, hogy ezek a pontok szerencsére lokális ablakokban is detektálhatók, amely jelentősen megkönnyíti ennek a műveletnek az elvégzését.

6.3.3. Objektumok címkézése és számlálása

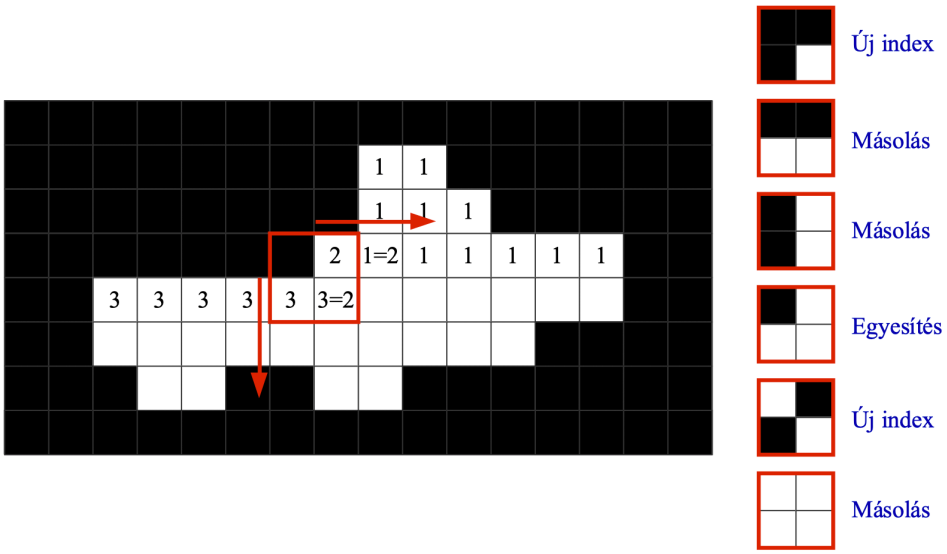
Az első lényeges, bináris képeken végzendő algoritmus az objektumok számlálása és felcímkézése. Ennek az eljárásnak a célja az, hogy az egymástól elkülönülő objektumokat mind egyedi címkével lássuk el, majd az címkézés befejezésekor a legnagyobb címke sorszámából megkapjuk az objektumok számát. A címkézésre alapvetően két elterjedt módszer létezik: a rekurzív, illetve a szekvenciális módszer.

A rekurzív módszer egy rendkívül egyszerű algoritmus. A lépései a következők:

1. Az első címkézetlen „1” pixel megkeresése, és L címkével való megjelölése.
2. A pixel összes „1” értékű szomszédjának L címkével való megjelölése, és a 2. lépés meghívása az összes szomszédra.
   (a) Ha nincs több jelöletlen „1” értékű szomszéd, akkor leáll az algoritmus.
3. Ugrás az 1. pontba és az L inkrementálása.

A módszer lépései rendkívül egyszerűek, azonban nagy, összefüggő objektumok esetén a rekurzió rendkívül mélyre mehet, ami problémákat eredményezhet különösen kis teljesítményű feldolgozó eszközök esetében. Ilyen esetekben célszerű a bonyolultabb, szekvenciális módszert alkalmazni. Ez az algoritmus a kép pixelein sorban halad végig, és minden új „1” értékű képpont esetén az alábbi szabályrendszer alapján ad új értéket a pixelnek:

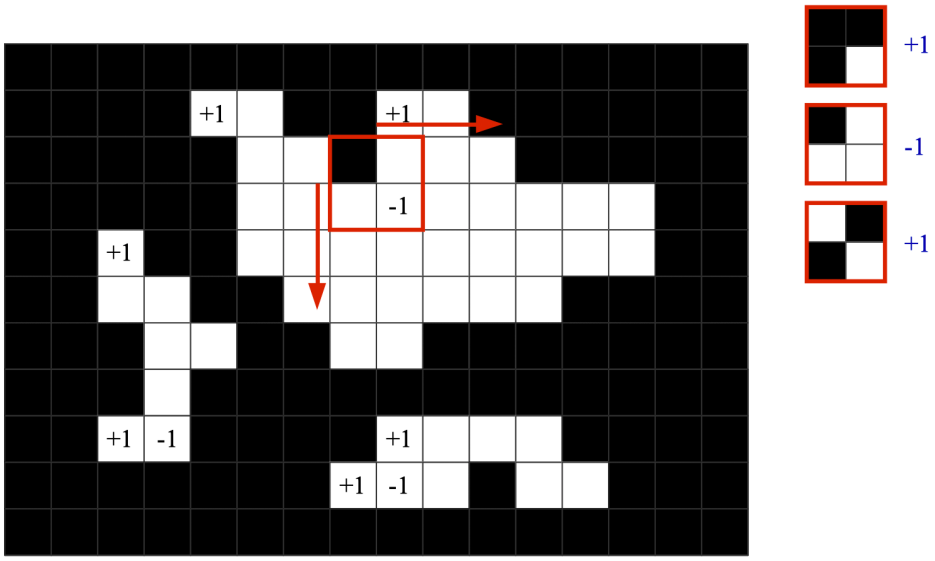
- Ha a képpontnak csak a felső vagy csak a bal oldali szomszédja címkézett, akkor a jelenlegi pixel az ő címkéjüket kapja.
- Ha a felső és a bal oldali szomszéd ugyanazt a címkét viseli, akkor szintén ugyanazt a címkét kapja.
- Ha a felső és a bal oldali szomszéd különböző címkét visel, akkor a pixel a felső képpont címkéjét kapja, valamint egy külön tárolóba feljegyezzük a két címkeérték egyenlőségét.
- Ha a képpontnak nincsen címkézett szomszédja, akkor új címkét vezetünk be a számára.



6.8. ábra. *Az objektum címkézés szekvenciális algoritmusának elve.*

A szekvenciális algoritmus futásának végén még egyszer végig kell haladnunk a képen, hogy a futás közben feljegyzett, ugyanahhoz az objektumhoz tartozó címkék helyett egy közös címkét adjunk az objektumoknak. Érdemes megjegyezni, hogy ez az algoritmus felhasználható objektumok számlálására is. Ekkor az objektumok számát mindig növeljük akkor, amikor az előző verziónál új címkét vennénk fel, és csökkentjük akkor, amikor címkéket olvasztanánk össze.

Alkalmazás: A bináris képek feldolgozásának egyik szemléletes alkalmazása a készpénzérmék automatikus számlálása, amely az ilyen fizetőeszközöket alkalmazó tranzakciókat képes gyorsítani. Ehhez célszerű egy külön berendezést készíteni, ahol egy meghatározott színű lapra lehet az érméket lehelyezni, amit egy ismert távolságra lévő számítógéphez kapcsolt kamera figyel. A meghatározott háttér miatt a kép küszöbözéssel könnyedén binárissá tehető úgy, hogy azon az érmék lesznek „1” értékkel jelölve. Az érmék különállóságát és lyukmentességét egymás után végzett nyitás-



6.9. ábra. *Az objektumszámlálás szekvenciális algoritmusának elve.*

és zárásműveletekkel biztosíthatjuk. Ezt követően címkézés segítségével különválasztjuk az egyes objektumokat, amelyeknek a területe alapján következtetünk az érme címletére. Természetesen az érmék valódiságát szükséges mintaillesztéssel is ellenőrizni.

6.4. Objektum tulajdonságok

Amennyiben sikerült elkülönítenünk a különböző bináris objektumokat, akkor a következő fontos lépés az, hogy az egyes objektumokhoz különböző jellemzőket határozzunk meg, melynek segítségével az objektumok jellemezhetők és azonosíthatók. Ezekhez gyakorta felhasználhatjuk az egyes objektumok vetületeit. A vetület egy olyan tömör reprezentáció, ahol a kép két tengelye mentén összeszámoljuk az egyes oszlopokban/sorokban lévő ”1” pixelek számát. Érdemes megjegyezni, hogy az objektum eredeti formája a több vetületből visszaállítható, az orvoslásban használt tomográfiás eljárások is ezen az elven működnek.

6.4.1. Pozíció, orientáció

Az egyes objektumoknak rendkívül fontos leíró mennyisége még a nyomaték, avagy idegen kifejezéssel a momentum. Nyomatékból számos rend létezik, amelyek közül számunkra a nullad-, illetve az elsőrendű nyomatékok hasznosak. A nulladrendű nyomaték egész egyszerűen a pixelintenzitások összege. Mivel itt bináris képeket tárgyalunk, így az itteni objektumok nulladrendű nyomatéka azok területét fogja megadni. Az objektumok területét és tömegközéppontját pedig meghatározhatjuk az első- és a nulladrendű nyomatékok segítségével az alábbi képlet alapján:

$$M_{00} = \sum_{x=0}^{W}\sum_{y=0}^{H} I(x,y); \quad M_{10} = \sum_{x=0}^{W}\sum_{y=0}^{H} x * I(x,y); \quad M_{01} = \sum_{x=0}^{W}\sum_{y=0}^{H} y * I(x,y)$$
$$A = M_{00}; \quad C = \left(\frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}}\right) \tag{6.1}$$

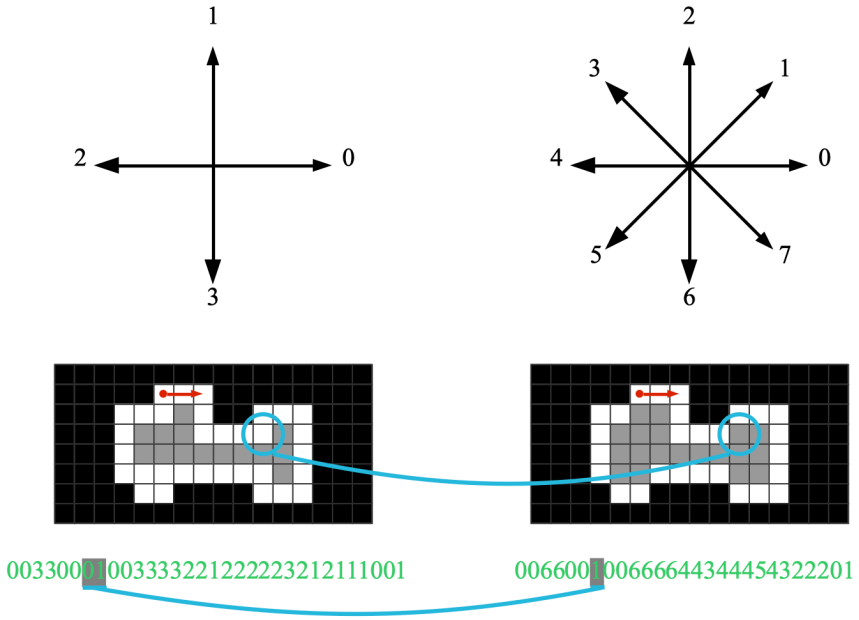
A nyomatékok felhasználhatók az objektum orientációjának meghatározására is, konkrétan a legkisebb nyomatékkal rendelkező tengelyt kell megkeresni, amely a másodrendű nyomatékok segítségével egyértelműen meghatározható, amennyiben az objektum nem szimmetrikus (szimmetrikus objektum orientációja nem egyértelmű). Az orientáció meghatározható az alábbi módon:

$$M_{20} = \sum_{x=0}^{W}\sum_{y=0}^{H} x^2 * I(x,y); \quad M_{02} = \sum_{x=0}^{W}\sum_{y=0}^{H} y^2 * I(x,y); \quad M_{11} = \sum_{x=0}^{W}\sum_{y=0}^{H} xy * I(x,y)$$
$$M_x = M_{20} - \frac{M_{10}^2}{A}; \quad M_y = M_{02} - \frac{M_{01}^2}{A}; \quad M_{xy} = M_{11} - \frac{M_{10}M_{01}}{A}; \tag{6.2}$$
$$\Theta = atan\left(\frac{M_x - M_y + \sqrt{(M_x - M_y)^2 + 4M_{xy}^2}}{2M_{xy}}\right)$$

Érdemes megjegyezni, hogy létezik számos más mérőszám az objektumok orientációjára, amelyek általában egyszerűbben számolhatók. Amennyiben ismert az objektum befoglaló téglalapja, akkor ennek arányai és méretei felhasználhatók az orientáció jellemzésére. Ugyanígy meghatározhatjuk az objektumon belüli legnagyobb távolság, vagy a tömegközépponttól vett legnagyobb távolság irányát is.

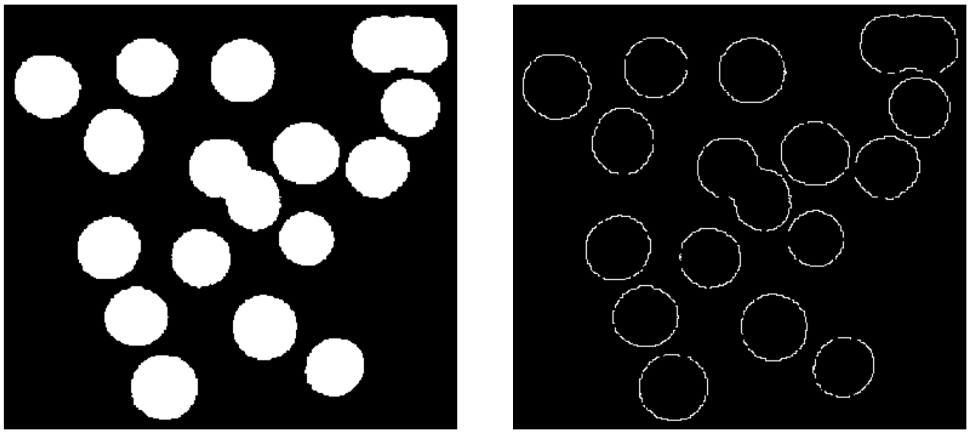
6.4.2. További mércék

Bináris objektumoknak további mércéi lehetnek. Ezek közül az egyik első az objektumok alakjának, formájának leírása. Ennek egyik módszere a lánckód alapú határábrázolás. A lánckód használata során egy sorszámot rendelünk minden irányhoz [0–3] vagy [0–7] között, a szomszédossági konvenció függvényében. Ezt követően egy adott kiindulási pontból sorban végig haladunk az objektum határának összes pontján, minden lépésnél feljegyezve annak irányát. A kiindulási pontba visszaérve megkapjuk az objektum körvonalának egy leíróját.



6.10. ábra. *A lánckód előállításának módszere.*

A lánckód felhasználható az objektum kerületének meghatározására, azonban ennek használatánál figyelembe kell venni azt, hogy (különösen 4 szomszédosság esetén) az objektum körvonalán cikkcakk módon haladtunk végig, ami mesterségesen megnöveli a kerület hosszát. Éppen ezért az egyes lépések euklideszi távolságát felhasználva sokkal pontosabb becslést kaphatunk. Érdemes megjegyezni, hogy a korábban ismertetett műveleteket felhasználhatjuk arra, hogy előállítsunk egy olyan bináris képet, ahol csak az objektumok határvonalai vesznek fel „1” értéket. Ehhez a képen



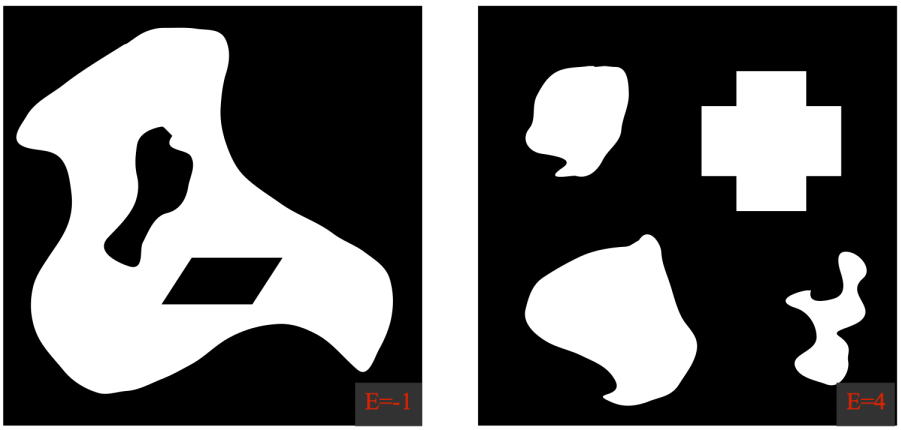
6.11. ábra. *Bináris objektumok kontúrjai.*

eróziót kell végeznünk, majd az így keletkezett és az eredeti képet egymásból kivonva megkapjuk az úgynevezett kontúrképet.

Amennyiben egy objektum kerülete, területe és tömegközéppontja rendelkezésünkre áll, még számos más alakleírót is hozzárendelhetünk az egyes objektumokhoz, amelyek segítségével azonosíthatók, osztályozhatók lesznek. Ilyen leíróból számos létezik; ezekre az egyik legegyszerűbb példa az objektum kerületének és területének az aránya.

Elterjedt megoldás még a kontúr lenyomatának használata, amelyet úgy számolhatunk ki, hogy egy bizonyos irányból elindulva minden irányban megmérjük az objektum középpontjának és a határvonalának a távolságát, és az így kapott függvényt használjuk az objektum alakjának leírójaként. A kezdőpontot általában úgy választjuk meg, hogy következetesen a legnagyobb távolságú irányból indulunk ki, így elforgatás esetén is ugyanazt a leírót kapjuk. Amennyiben a kapott leíró függvényt normáljuk, akkor a skálázásra is invariáns leírót kaphatunk.

Rendkívül elterjedt leíró mérték még az Euler-szám, amely az egy objektumban megtalálható összefüggő régiók számának és az objektumon belül található lyukak számának a különbsége. Ez a mérték rendkívül jól használható olyan esetekben, amikor az objektumok formája számottevő torzulásnak lehet kitéve, azonban a legtöbb gyakorta előforduló geometriai torzítás ezt a tulajdonságot nem változtatja meg, így a különböző Euler-számú objektumok megkülönböztethetők maradnak



6.12. ábra. *Az Euler szám kiszámolása.*

További Olvasnivaló

[1] R. Szeliski, *Computer Vision*. Springer London, 2011. DOI: 10.1007/978-1-84882-935-0. cím: https://doi.org/10.1007%2F978-1-84882-935-0.

[2] L. Ross, "The Image Processing Handbook, Sixth Edition, John C. Russ. CRC Press, Boca Raton FL, 2011, 972 pages. ISBN 1-4398-4045-0(Hardcover)", *Microscopy and Microanalysis*, 17. évf., 5. sz., 843–843. old., 2011. szept. DOI: 10.1017/s1431927611012050. cím: https://doi.org/10.1017%2Fs1431927611012050.

[11] R. O. Duda és P. E. Hart, "Use of the Hough transformation to detect lines and curves in pictures", *Communications of the ACM*, 15. évf., 1. sz., 11–15. old., 1972. jan. DOI: 10.1145/361237.361242. cím: https://doi.org/10.1145%2F361237.361242.

7. fejezet

Döntéshozás

Az eddigiekben számos módszert ismertünk meg arra, hogy a képek minőségét javítsuk, illetve ezek után a pixel információkból valamilyen magasabb szintű, absztrackt leírókat generáljunk, amelyek a képi tartalmakat jól leírják. A hagyományos látórendszerek utolsó algoritmikus lépése az úgynevezett döntéshozás, amikor ezek alapján a leírók alapján döntéseket kell hozni. A korábbiakban bizonyos feladatok esetében már érintettük ezt a témát (pl. SIFT jellemzők párosításának módja), azonban általános megoldást nem adtunk erre a kérdésre. Ennek oka, hogy az itt alkalmazott módszerek már a hagyományos látórendszerek esetében is gyakran a gépi tanulás módszertanára épülnek, melyet az alábbiakban ismertetünk.

7.1. Gépi tanulás

A számítógépes látás területének alapvető célja magas szintű információk kinyerése a képekből. Azonban a bevezető előadásban ismertetett problémák ez meglehetősen nehézzé tehetik. Emiatt adja magát a felvetés, hogy az emberi intelligencia képességeit próbáljuk meg valamilyen módon a számítógépes látás módszereibe beleültetni, ez által lehetővé téve a problémák megoldását. A mesterséges intelligencia tudományterülete hatalmas, számos lehetséges algoritmus áll rendelkezésünkre. Ezen algoritmusok jelentős része egzakt algoritmus, vagyis könnyen megfogalmazható utasítások és logai feltételek valamilyen sorozataként. Ezek az algoritmusok voltaképpen a készítő intelligenciáját aknázzák ki az intelligens működés eléréséhez. Az emberi látás működésének megértése híján ezek az algoritmusok nem segítenének megoldani a fent felsorolt problémákat.

A mesterséges intelligencia módszereinek létezik azonban egy másik csoportja, ezek az úgynevezett tanuló algoritmusok. A tanuló eljárások a probléma megoldására egy általános, paraméterezhető modellt nyújtanak, és a tanulás folyamata során egy tanító adathalmazt használnak fel arra, hogy ezeket a paramétereket olyan módon határozzák meg, hogy a kezdeti, általános modell az adott probléma megoldására specializálódjon. Ezen algoritmusok hatalmas előnye, hogy segítségükkel megoldhatunk olyan problémákat is, amelyek megoldásának módszerét magunk nem ismerjük, feltéve, hogy képesek vagyunk előállítani egy az algoritmus tanításához megfelelő adatbázist.

Fontos hátránya azonban a gépi tanulás módszereinek, hogy a tanítás végén kapott modell általában fekete doboz jellegű, vagyis a kapott modell megvizsgálásával nem feltétlenül jutunk közelebb a probléma megoldásának megértéséhez. A fekete doboz jelleg miatt azonban rendkívül nehéz megérteni az esetleges hibák, tévesztések okát, és jövőbeli elkerülésüknek a módját. Fontos még megjegyezni, hogy a gépi tanulás módszerei a paramétereket a tanulás során általában statisztikai módszerekkel, vagy numerikus optimalizálás segítségével határozzák meg, következésképp a helyes működésükre nem lehet garanciát mondani. Ezen hátrányok ellenére a számítógépes látás és általánosságban az érzékelés területén toronymagasan felülmúlják a hagyományos algoritmusok teljesítményét.

7.2. Tanuló algoritmusok felépítése

A gépi tanulás során egységesen egy tanuló algoritmus bemenetét x, kimenetét y, paramétereit pedig ϑ jelöli. Ezekkel a jelölésekkel egy tanuló algoritmus modellje megadható egy paraméterezett függvény formájában:

$$\hat{y} = f(x, \vartheta) \tag{7.1}$$

Minden tanító algoritmushoz tartozik egy költségfüggvény (gyakran nevezik még hiba- vagy veszteségfüggvénynek), amely a tanuló algoritmus kimenetéhez hozzárendel egy hiba értéket, melynek segítségével az algoritmus teljesítményét tudjuk értékelni. Ezen felül minden tanító algoritmusnak része egy vagy több optimalizálási módszer is, amelynek segítségével a hibafüggvényt minimalizálhatjuk a paraméterek változtatásával. A legtöbb tanító algoritmushoz tartoznak még úgynevezett hiperparaméterek is, melyek olyan paraméterek, amelyek a megoldás minőségét általában befolyásolják, azonban nem tudjuk őket az optimalizálási módszerrel meghatározni. Tipikusan magának az optimalizálási módszernek, vagy a modell struktúrájának tulajdonságai ilyenek.

7.2.1. Tanulás típusai

A gépi tanulás módszereit számos fontos szempont szerint lehetséges csoportosítani, melyek közül az első az algoritmus kimenete szerinti csoportosítás. Regresszió esetén a tanuló rendszer kimenete egy folytonos szám. Amennyiben az algoritmus kimenete egy bináris változó, vagy egy véges halmazból származó egész szám, akkor pedig osztályozásról beszélhetünk. Az osztályozás alapeset a bináris osztályozás, mivel egy többértékű osztályozó rendszer előállítható több bináris osztályozó kompozíciójaként. Ez elképzelhető úgy, hogy minden osztályhoz tartozik egy bináris osztályozó, amely az adott osztályt minden mástól meg tudja különböztetni, és a végső osztályt az egyes osztályozók konfidenciája dönti el. Elképzelhető olyan rendszer is, ahol az egyes osztályozók két osztály között tudnak dönteni, a végső osztályt pedig a sportbajnokságokhoz hasonló pontozási módszerrel döntik el.

Egy másik fontos csoportosítási elv a tanításhoz felhasznált tanító adatok milyensége. A gépi tanulás legegyszerűbb formája a felügyelt tanulás. Ebben az esetben a tanító adatok bemenet-elvárt kimenet párokban állnak rendelkezésre, vagyis minden bemenetre ismerjük a helyes választ, a tanuló algoritmustól pedig azt várjuk el, hogy ezeket minél nagyobb arányban, vagy minél pontosabban találja el. Előfordulhat, hogy a tanító adatbázis csak egy részéhez áll rendelkezésünkre az elvárt kimenet, ebben az esetben félig felügyelt tanulásról beszélhetünk.

A felügyelt tanulásnak azonban jelentős korlátai vannak. Egyrészt, a felcímkézett tanító adatbázisok előállítása rendkívül hosszadalmas és drága feladat. Másrészt a címkézés minősége alapvetően korlátozza a tanuló algoritmus minőségét. Végül pedig ahogy az egzakt algoritmusok használata csak akkor lehetséges, ha tudatos szinten értjük az adott probléma megoldását, a felügyelt tanuló algoritmusok pedig csak akkor használhatók, ha mi magunk meg tudjuk oldani az adott feladatot. Ebből következik, hogy egy felügyelt tanuló algoritmus sosem fog tudni olyan feladatokat megoldani, amit mi nem.

Létezik azonban felügyelet nélküli tanulás, amikor a tanító adathalmaz csak bemeneti értékekből áll, az elvárt kimenetet egyáltalán nem ismerjük. Ilyen adatbázisokat rendkívül olcsó előállítani, mivel a legtöbb esetben az új adatok beszerzése automatizálható. Ilyen esetekben azt várjuk el a tanuló algoritmustól, hogy képes legyen valamilyen belső struktúrát találni az adathalmazban, és ezáltal azt kompakt módon leírni. Az algoritmusok célja, hogy a bemeneten látott adatokat valamilyen kompakt modell segítségével magyarázzák, azoknak a belső struktúráját feltérképezzék. Ilyen algoritmusokra jó példák a korábbi fejezetben ismertetett klaszterezési eljárások (k-Means, MoG), amik voltaképpen az osztályozás felügyelet nélküli változatának tekinthetők. A későbbiekben ismertetett TLS módszer is értelmezhető a lineáris regresszió felügyelet nélküli változataként.

A gépi tanulás harmadik fő fajtája a megerősítéses tanulás, ami két fontos dologban különbözik a másik két típustól. Egyrészt a megerősítéses tanulás esetén szinte mindig összefüggő döntések

sorát kell az algoritmusnak meghozni, de a döntéssorozat helyességéről jellemzően nem kap minden döntés után visszajelzést. Másfelől a kapott visszajelzés során az algoritmus csak egy értékelést kap a döntések minőségéről, azt nem tudja meg, hogy a helyes döntés mi lett volna. A megerősítéses tanulási feladatokra tipikusan jó példák a különböző játékok (pl. sakk, go, számítógépes játékok) valamint a különböző járműirányítási feladatok.

7.2.2. Nehézségek

Első ránézésre a gépi tanulás könnyedén tűnhet egyfajta varázslatos módszernek, amivel a világ összes problémáját könnyedén meg lehet oldani. A gyakorlatban azonban ezeknek a módszereknek is bőségesen akadnak limitációik és csapdáik, amikbe könnyedén bele lehet esni. A csapdák elkerülésének érdekében fontos mindig emlékezni arra, hogy ezek az algoritmusok tanító adatokból dolgoznak, ami azt jelenti, hogy a világnak csak azt a szegletét tudják értelmezni, amit a tanító adatok lefednek. Ez azt jelenti, hogy a felhasznált tanító adatainknak minden lehetőséget le kell fednie, mivel nem tudjuk megjósolni, hogy az algoritmus hogyan fog még soha nem látott esetekben viselkedni.

Érdemes azt is észben tartani, hogy például a tanuló látórendszerek tipikusan nem tudnak olyan összefüggéseket megtanulni, amelyhez a mozgás képessége, vagy más érzékszervek szükségesek. Ez persze így leírva triviálisnak tűnhet, de gondoljunk csak a szék fogalmára: „egy olyan tárgy, amire rá lehet ülni”. Ezt a definíciót pedig fel tudom használni a szék felismerésére, hiszen ránézésre általában el tudjuk dönteni, hogy valamire rá lehet-e ülni. Egy olyan algoritmus, ami viszont nem tud mozogni, csak összefüggéstelen képeket kap, sosem fogja az ülés fogalmát megalkotni.

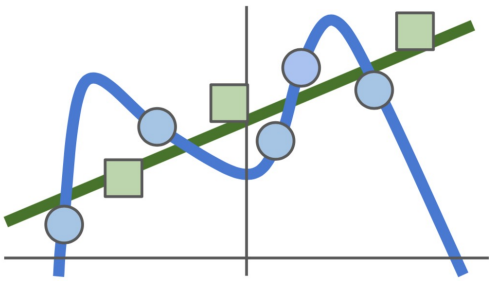
Egy másik gyakori csapdája a gépi tanulásnak a tanuló eljárás komplexitásának kérdése. Alapvető emberi intuíció ugyanis az, hogy ha a tanuló algoritmus nem elég pontos, akkor, ha komplexebbé („okosabbá”) tesszük, akkor a teljesítmény növelhető. Egy modell komplexitását számos módon lehet növelni: a bemeneti változók és a paraméterek számának növelése két nyilvánvaló módszer. Ezen felül a tanuló módszer hiperparaméterei is általában befolyásolják a komplexitást. A modell komplexitásának növelése azonban kétélű fegyver: alapvetően a szituációtól függ, hogy ront, vagy javít-e a helyzeten.

Előfordulhat olyan eset, amikor az algoritmus nem elég komplex az adott feladat megoldásához. Ebben az esetben azt tapasztaljuk, hogy a tanító adatbázison elért hiba meglehetősen nagy, és ha az algoritmust ezután olyan új adatokon teszteljük, amiket a tanítás során nem látott, akkor hasonlóan hagy hibát kapunk. Ezt a jelenséget alulillesztésnek (underfitting) hívjuk. Ebben az esetben a komplexitást növelve a tanítási és a tesztelési hiba is egyre csökken. Egy idő után azonban azt fogjuk tapasztalni, hogy a tanítási hiba csökkenése mellett a tesztelési hiba először stagnálni, majd növekedni kezd a komplexitás növelésével. Ezt a jelenséget hívjuk túlillesztésnek (overfitting).

A túltanulás oka az, hogy a tanításra használt adathalmaz nem teljesen tökéletes. Egyrészt véges, ami azt jelenti, hogy az algoritmus feladata, hogy megtanuljon általánosítani az adathalmaz segítségével. Másrészt mind a bemenetek, mind a kimenetek zajjal terheltek, így az algoritmus tanítási hibája tökéletes általánosítás esetén sem lesz nulla. Ez azt jelenti, hogy egy idő után az algoritmus már csak úgy tudja tovább csökkenteni a tanítási hibát, ha elkezdi egyesével memorizálni a tanító adatokra adandó helyes választ. Ennek következtében egy a problémát általánosságban megoldó algoritmus helyett egyre inkább egy asszociatív memóriára kezd hasonlítani. Ez azt jelenti, hogy a tanító adatbázisban nem szereplő bemenetekre egyre rosszabb válaszokat ad. Ráadásul minél komplexebb az algoritmus, annál könnyebben tud egy nagy adatbázist memorizálni.

7.3. Képosztályozás

A számítógépes látás egyik legegyszerűbb formája a már korábban ismertetett osztályozás, vagyis amikor egy képhez egyetlen címkét rendelünk, amely a képen található objektum kategóriáját kódolja. A gyakorlatban egy osztályozást végző számítógépeslátás-megoldás számos algoritmus

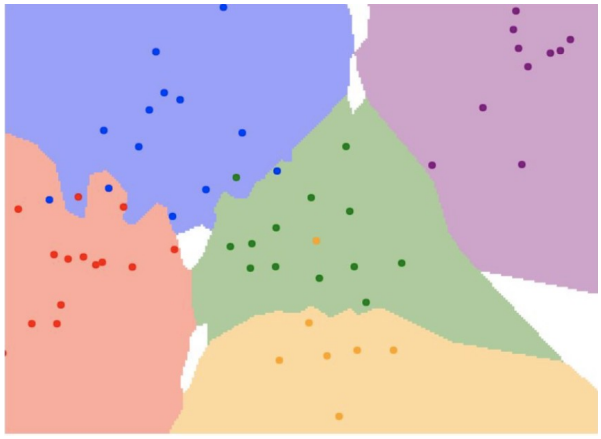


7.1. ábra. *Az overfitting egy dimenzió esetén. Látható, hogy a kék színű modell rátanul a tanító adatbázisban lévő zajra, így a tesztadatokon (zöld) rosszul teljesít.*

egymás után történő végrehajtásából áll, amelyet algoritmikus csővezetéknek (pipeline) nevezünk. Ezek első lépése a képek készítése és digitalizálása, amelyet egy előfeldolgozó, képjavító (zajszűrések, intenzitástranszformációk) blokk követ. Ezt követően egy jellemző kiemelési fázis következik, amelynek célja, hogy a képen fellelhető információt a pixelintenzitások által meghatározott térből egy ennél nagyobb absztrakciós szinten létező képjellemzők által meghatározott térbe transzformálja. Ezeket a képjellemzőket úgy tervezzük meg, hogy az általuk meghatározott térben könnyen elválaszthassuk a feladat szempontjából releváns információkat a zavaró hatásoktól. Az utolsó lépés egy döntési fázis, amelyben az algoritmus a képjellemzők alapján címkét rendel az adott képhez.

7.3.1. Legközelebbi szomszéd

Érdemes a képjellemzők szükségességét egy szemléletes példán keresztül demonstrálni. Lehetséges ugyanis képosztályozást tisztán intenzitás vagy szín alapján végezni, legegyszerűbben például a k legközelebbi szomszéd elnevezésű, vagyis a kNN (az angol k Nearest Neighbours kifejezésből) algoritmus segítségével. Ennek a végtelenül egyszerű eljárásnak a lényege, hogy egy már ismert címkéjű képekből álló adatbázisban megkeresi az éppen osztályozandó képek k darab legközelebbi szomszédját. Ezek a szomszédok aztán többségi elven, szavazással döntik el az új kép címkéjét. A kNN a képek távolságát általában a két kép pixeleinek abszolút vagy négyzetes különbségeinek összegeként definiálja. A módszerben felhasznált k változó értékét a tervező szabadon választhatja.



7.2. ábra. *A kNN algoritmus döntési területei $k = 3$ esetében.*

A megoldás alapvető problémája, hogy az intenzitás és színértékek közti különbségek összege nincs összhangban a képek hasonlóságával, különösen nem a szemantikus osztályok közti különbséggel. Könnyen belátható, hogy a különböző megvilágítással vagy háttér előtt készült képek különbsége jelentős lesz a rajtuk szereplő objektum osztályától függetlenül. A helyzet különösen rossz olyan objektumok esetén, amelyek hajlamosak számos különböző színezetben előfordulni (például állatok, járművek, ember). E probléma miatt a színinformációt csak olyan esetekben használjuk képek

osztályozására, amikor mind az objektumok kinézetét, mind a környezet vizuális tulajdonságait kézben tudjuk tartani. Ilyen szituációkra jó példák a különböző beltéri ipari alkalmazások (például alkatrész-felismerés) vagy a virtuális- és kiterjesztettvalóság-rendszerek.

7.3. ábra. *A fenti módosított képek négyzetes értelemben megegyező távolságra vannak az eredetitől (bal fent).*

7.3.2. Lineáris regresszió

Egy másik alapvető paraméterbecslő algoritmus a lineáris regresszió, vagy lineáris legkisebb négyzetes becslés (LS) módszere. A módszer alapelve, hogy a tanító adathalmazban lévő bemenetek és az előírt kimenet közötti összefüggést egy lineáris egyenlettel közelítjük. Ez szemléletesen azt jelenti, hogy az adathalmazra egy olyan egyenest (több dimenzióban hipersíkot) próbálunk illeszteni, amely a lehető legkisebb négyzetes hibával közelíti az adatpontokat. A lineáris regresszió modellje az alábbi:

$$X\vartheta = Y \tag{7.2}$$

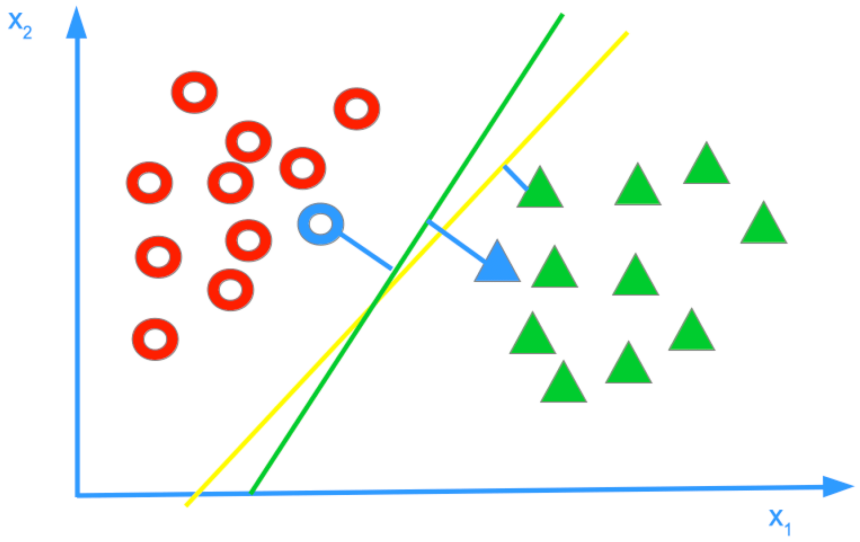
Ahol az X mátrix minden sora egy tanító adat, a ϑ vektor a hipersík paramétereit tartalmazza, Y pedig az elvárt kimenetek vektora. Az egyenlet hibáját minimalizáló megoldáshoz optimalizáló eljárás nem szükséges, ugyanis az optimális megoldás zárt alakban előállítható. Érdemes megjegyezni, hogy az X mátrix oszlopaiban nem csak különböző változók, hanem akár egyetlen változó különböző hatványai is szerepelhetnek. Ekkor nem egyenest, hanem valamilyen polinomot illesztünk az adathalmazra, mely esetben polinomiális regresszióról beszélhetünk.

$$\begin{aligned} ||E||^2 = (Y - X\vartheta)^T(Y - X\vartheta) &= Y^TY - 2\vartheta^TX^TY + \vartheta^TX^TX\vartheta \\ \frac{\partial||E||^2}{\partial\vartheta} &= -2X^TY + 2X^TX\vartheta := 0 \\ \hat{\vartheta}_{LS} &= (X^TX)^{-1}X^TY \end{aligned} \tag{7.3}$$

7.3.3. SVM

A lineáris megközelítést az osztályozás módszereire is ki lehet terjeszteni, ezt azonban úgy lehet elképzelni, hogy az egyes osztályokat egy egyenessel (hipersíkkal) próbáljuk meg egymástól elválasztani. Könnyen belátható azonban, hogy általában végtelen sok ilyen sík létezik, így valamilyen

módon jó lenne ezek közül a legjobbat kiválasztani. Ezt elvégezhetjük úgy, hogy megpróbáljuk azt a hipersíkot megtalálni, amelyik a legnagyobb biztonsággal választja el a két osztályt, azaz a síkhoz legközelebbi adatpont a lehető legmesszebb legyen az elválasztó hipersíktól. A síkhoz legközelebbi tanító adatot szupport vektornak, míg annak a síktól való távolságát résnek (margin) nevezzük.



7.4. ábra. *A bináris osztályozási feladat: A két osztály, az elválasztó hipersík, a support vektorok és a margin.*

A számítógépes látásban népszerű algoritmus az úgynevezett SVM (Support Vector Machine) algoritmus, amely az osztályokat maximális réssel elválasztó hipersíkot keresi meg. Az SVM döntésfüggvénye az alábbi módon írható fel:

$$\hat{y}(x) = \sum_{i}^{N} \alpha_i y_i K(x_i, x) \tag{7.4}$$

Ahol N a tanító adatok száma, x_i és y_i az i-edik tanító adat be- és kimenete, α_i az i-edik tanító adathoz az SVM által megtanult súly, míg K egy kernel függvény. Ez a kernel függvény egy hasonlósági mérce, az éppen osztályozandó bemenet és a tanító adatok között. Ez tulajdonképpen azt jelenti, hogy minden tanító adat kimenete olyan mértékben befolyásolja a döntést, amennyire a két tanító adat hasonlít egymásra. Az ilyen jellegű módszereket általánosságban kernel módszereknek nevezzük, és bizonyos tekintetben a legközelebbi szomszéd módszer általánosításának tekinthetők.

Az SVM módszer egy fontos tulajdonsága, hogy a tanulás során előálló együtthatók csak a szupport vektorok esetén térnek el nullától, vagyis a fenti összeget elég csak erre a néhány vektorra elvégezni. Fontos még a kernel függvény megválasztásáról is beszélni, ugyanis ez alapvetően befolyásolja az SVM képességeit. Az alapértelmezett kernel függvény a lineáris kernel, ami a két bemeneti vektor skaláris szorzatát számolja ki. Lineáris kernel használata esetén az SVM a korábban bevezetett maximális résű elválasztó hipersíkot adja meg.

Érdemes azonban belátni, hogy nagyon sok valódi probléma esetén egyáltalán nem létezik olyan lineáris felület, ami elválasztaná a két osztályt. Erre egy kiváló példa a rendkívül egyszerű kizáró vagy probléma, de számos más szemléletes példa akad ilyen osztályozási feladatokra. Az SVM rendkívül hasznos tulajdonsága, hogy nemlineáris kernel függvények használata esetén képes az egyes osztályokat nemlineáris görbék segítségével is elválasztani, vagyis a módszer könnyedén kiterjeszthető. A két leggyakrabban használt nemlineáris kernel függvény a polinomiális, és az RBF (radiális bázisfüggvény).

$$\begin{aligned} K_{Lin}(x_1, x_2) &= x_1^T x_2 \\ K_{Poly}(x_1, x_2) &= (x_1^T x_2 + c)^k \\ K_{RBF}(x_1, x_2) &= e^{-\gamma ||x_1 - x_2||^2} \end{aligned} \tag{7.5}$$

Ahol c, k és γ a módszer komplexitását kontrolláló paraméterek. A kernel függvények közös tulajdonsága, hogy mind szimmetrikus és pozitív szemidefinit függvények. Valóságban a lineáris kernelhez hasonlóan minden kernel függvény a két bemenete között egy skaláris szorzatot számol ki, azonban előtte a bemenetei vektorokon valamilyen nemlineáris transzformációt hajt végre (természetesen mindezt implicit módon). Ily módon az SVM tulajdonképpen minden kernel esetében hipersíkot illeszt, csak éppenséggel a paramétertér, amiben ezt teszi az eredeti paraméterek nemlineáris transzformációjából képződött, így az illesztett hipersík az eredeti térben valójában egy görbe lesz.

7.4. Nem felügyelt tanulás

A fent ismertetett módszerek eddig mind a felügyelt tanulás területéről kerültek ki. Azonban, ahogy azt korábban is említettük, léteznek nem felügyelt algoritmusok is, amelyek ugyanezeket a feladatokat (osztályozás, regresszió) címkék nélküli adatbázisokon kísérlik elvégezni. Bár ez elsőre szinte lehetetlen feladatnak hangzik, gondoljunk egy pillanatra bele: Ismerünk-e olyan algoritmust, amely képes egy n-dimenziós térben pontokat úgy csoportokba rendezni (ergo osztályozni), hogy a csoportok számát és típusát sem ismerjük, de mégis szeretnénk kompakt, összetartozó csportokat kapni? Hát persze, hogy ismerünk, a korábbi fejezetben tárgyalt klaszterező módszerek pontosan ezt csinálják. Valóban, a klaszterezés felfogható, mint egy felügyelet nélküli tanuló osztályozó algoritmus.

7.4.1. Total Least Squares

Ha a Lineáris Regresszió az egyenesillesztés felügyelt változata, akkor ennek a felügyelet nélküli megfelelője az úgynevezett Total Least Squares (TLS) módszer. Ahogy azt a korábban említettük, a legkisebb négyzetek (LS) módszerének alapegyenlete az alábbi:

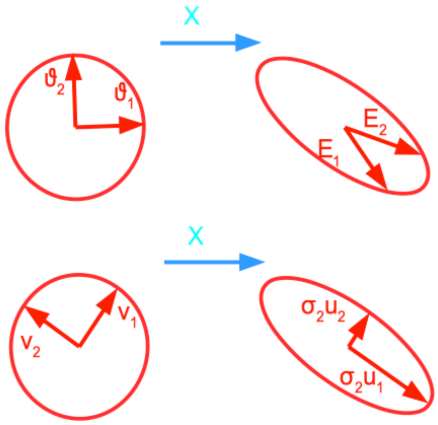
$$X\vartheta = Y \tag{7.6}$$

Ahol az X mátrix minden sora egy egyenlet ismert paramétere, a ϑ vektor az ismeretlen paramétereket tartalmazza, Y pedig az elvárt kimenetek vektora. Emlékeztetőül, a LS becslés az alábbi módon oldaható meg zárt alakban:

$$\begin{aligned} ||E||^2 = (Y - X\vartheta)^T (Y - X\vartheta) &= Y^T Y - 2\vartheta^T X^T Y + \vartheta^T X^T X \vartheta \\ \frac{\partial ||E||^2}{\partial \vartheta} &= -2X^T Y + 2X^T X\vartheta := 0 \\ \hat{\vartheta}_{LS} &= (X^T X)^{-1} X^T Y \end{aligned} \tag{7.7}$$

Felügyelet nélküli tanulás esetében azonban nincs kimenet, vagyis az Y vektor a nullvektor, amit totális legkisebb négyzetes (TLS) problémának nevezünk. Ekkor az ismeretlen paramétervektorra is előáll a triviális nulla megoldás, ami a legtöbb esetben számunka használhatatlan. Ennek a problémának a megoldása során előírjuk, hogy a megoldás normája legyen egy, hogy a triviális megoldást elkerülhessük. Ez szemléletesen azt jelenti, hogy az összes egység hosszú vektor közül azt keressük, amely az X mátrixszal való szorzás után a lehető legközelebb esik a nullához. Ez elképzelhető úgy, mint az X mátrix legkisebb „erősítésének" az iránya, amelyet pont az X legkisebb szinguláris értékéhez tartozó szinguláris vektor ad meg.

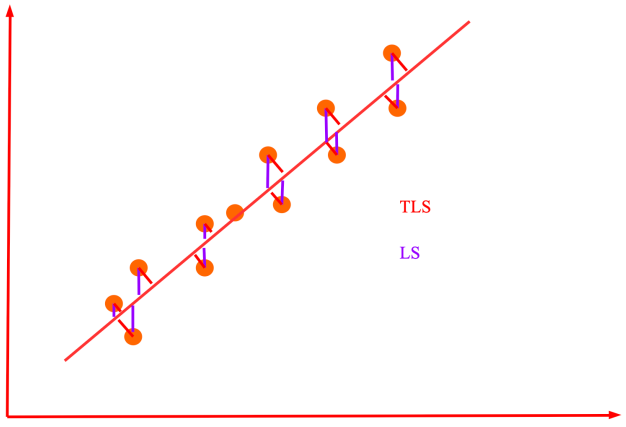
A szinguláris értékek és vektorok az alábbi módon definiálhatók:



7.5. ábra. *A szinguláris vektorok és értékek: A TLS algoritmus során az egységkör vektorai közül keressük azt, amelyik hossza a mátrixszorzás után a lehető legkisebb lesz. Ha veszünk az egységkörön belül egy ortogonális vektorrendszert, akkor ezek segítségével lineáris kombinációkkal előállítható az összes transzformált vektort tartalmazó ellipszoid (felül). Ha létezne egy olyan vektorrendszer, amely a mátrixtranszformáció előtt és után is ortogonális lenne, könnyen belátható, hogy ezek pont a kapott ellipszoid főtengelyeivel esnének egybe. Ezt a vektorrendszert hívjuk szinguláris vektoroknak, mely a sajátvektorok általánosításának tekinthetők.*

$$
\begin{gathered}
Xv_i = \sigma_i u_i; \quad X^T u_i = \sigma_i v_i \\
\sigma_i \geq 0; \quad v_i^T v_j = \delta_{ij}; \quad u_i^T u_j = \delta_{ij} \\
X = \begin{pmatrix} u_1 & u_2 & \dots & u_n \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_1 & \dots & 0 \\ \vdots & 0 & \dots & \vdots \\ 0 & 0 & \dots & \sigma_n \end{pmatrix} \begin{pmatrix} v_1 & v_2 & \dots & v_n \end{pmatrix}^T = U \Sigma V^T
\end{gathered}
\tag{7.8}
$$

Ahol σ_i a szinguláris érték, u_i és v_i pedig a hozzá tartozó jobb és bal szinguláris vektorok. δ_{ij} az úgynevezett Kroenecker-delta.



7.6. ábra. *Az LS és TLS módszerek által használt hibadefiníciók közti különbség. Látható, hogy az LS módszer hibája elszáll a végtelenbe függőleges egyenes esetén.*

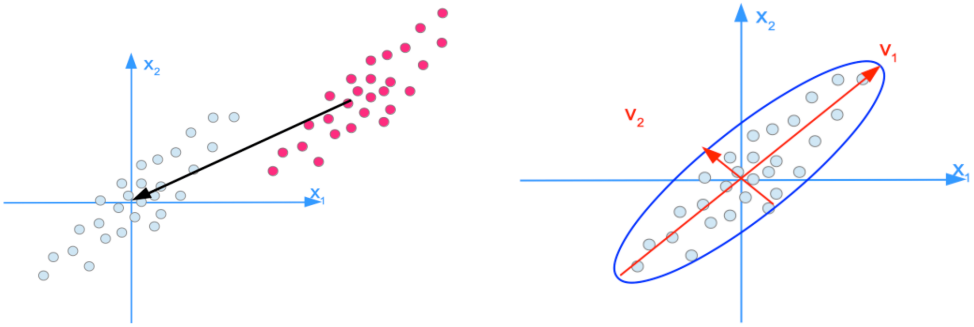
7.4.2. Dimenzió redukció

Egy másik gyakori feladata a nem felügyelt tanulásnak a dimenzióredukció. A dimenzióredukció során egy olyan bemeneti adathalmaz áll rendelkezésre, amelyben a változók dimenziószáma meg-

lehetősen magas, ezek a változók azonban nem feltétlenül függetlenek vagy relevánsak. Erre a képi adatok rendkívül jó példák, ugyanis ebben az esetben a változók száma nagy (minden pixel egy külön változó), a szomszédos pixelek között azonban erős összefüggések vannak, valamint számos pixel nem hordoz releváns információt számunkra (főleg a képek szélein fordul ez elő). Éppen ezért célszerű lenne az összefüggő változókat egybevonni, a haszontalanokat pedig eldobni, és ezáltal a feldolgozandó adatmennyiséget drasztikusan csökkenteni. Érdemes belátni, hogy egy ilyen eljárás képes lehet egy lokális képrészlet ideális tömör leírását megtalálni.

7.4.3. Főkomponens-analízis

A dimenzióredukció egyik legalapvetőbb módszere a főkomponens analízis. Ez az eljárás feltételezi, hogy az adathalmazunk normális eloszlású nulla középértékkel. Ez utóbbi könnyedén teljesíthető az adathalmaz változóinak várható értékének levonásával. A főkomponens analízis ezt követően az adathalmazt egy új ortogonális koordinátarendszerbe transzformálja, amelynek a bázisvektorait az adathalmaz kovarianciamátrixának ($\Sigma = (\mathbf{X} - \mu)^T(\mathbf{X} - \mu)$) sajátvektorai adják. Az így megkapott bázisvektorokat főkomponensnek nevezzük, és az ezek segítségével definiált új változók már statisztikailag függetlenek lesznek.



7.7. ábra. *A redukálandó adathalmaz normálása (bal), és a kapott főkomponensek (jobb).*

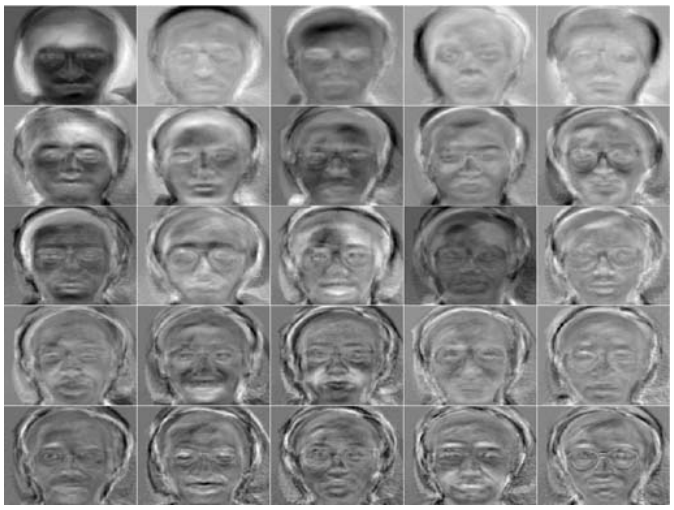
Ezt követően az így megkapott változók közül a legkisebb varianciájúakat eldobjuk egészen addig, amíg az adatbázis teljes varianciájának bizonyos százaléka alá nem estünk. Ez egyes főkomponensek varianciáját a sajátvektorhoz tartozó sajátértékek adják meg. Érdemes megjegyezni, hogy a főkomponens analízis normális eloszlású adatok esetére a lehető leghatékonyabb tömörítési eljárás. Érdemes még megjegyezni, hogy amennyiben az egyes változók eltérő mértékegységben szerepelnek, akkor érdemes ezeknek a skálázására odafigyelni, ez ugyanis a PCA eredményét torzíthatja.

Alkalmazás: A főkomponens analízis egy érdekes alkalmazása az eigenfaces, vagyis sajátarcok arcfelismerő rendszer, ahol egy emberi arcokból álló adatbázisból választották ki a legfontosabb sajátvektorokat. Ezek a sajátvektorok egyrészt felhasználhatók az emberi arcok különböző variációinak megértésére, valamint osztályozására is. Amennyiben egy kép közel esik az emberi arcok által kifeszített altérre, akkor arcszerűnek tekinthető, ellenkező esetben pedig nem.

A PCA-t és hasonló módszereket előszeretettel alkalmazzák még kisebb képrészletek tömör leírásához. Ebben az értelemben a PCA felfogható, mint képjellemző detektáló algoritmus, a képjellemzők leírását viszont bizonyos szempontból optimális módon teszi meg.

7.4.4. Lineáris Diszkriminancia-analízis

Érdemes még megjegyezni, hogy a főkomponens analízisnek létezik egy felügyelt tanuláshoz használható változata is, a lineáris diszkriminancia analízis (LDA). Ennek az eljárásnak a lényege, hogy az adathalmazhoz címkék is tartoznak, így két vagy több osztály található benne. Az LDA célja az, hogy a dimenziószámot úgy csökkentse, hogy az egyes osztályok szétválasztásának szempontjából használható információ maradjon meg. Ezt matematikailag úgy lehet megfogalmazni, hogy



7.8. ábra. *Egy arcokból álló képi adatbázis szignifikáns főkomponensei (vagyis az arcszerű képek). Érdemes megjegyezni, hogy ezek a képek a pixelek átlagának levonása után keletkeztek.*

az LDA nem a teljes adathalmaz varianciáját, hanem az osztályok közötti varianciát igyekszik maximalizálni.

7.5. Osztályozó és detektáló módszerek

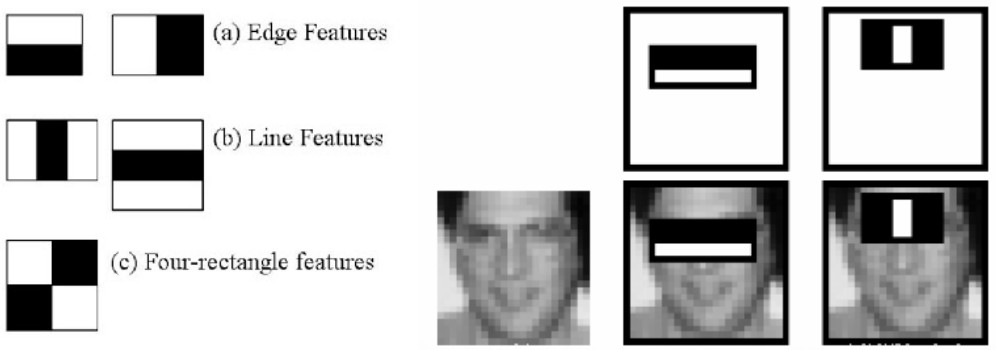
A számítógépes látás egyik legegyszerűbb formája a már korábban ismertetett osztályozás, vagyis amikor egy képhez egyetlen címkét rendelünk, amely a képen található objektum kategóriáját kódolja. A gyakorlatban egy osztályozást végző számítógépeslátás-megoldás számos algoritmus egymás után történő végrehajtásából áll, amelyet algoritmikus csővezetéknek (pipeline) nevezünk. Ezek első lépése a képek készítése és digitalizálása, amelyet egy előfeldolgozó, képjavító (zajszűrések, intenzitástranszformációk) blokk követ. Ezt követően egy jellemző kiemelési fázis következik, amelynek célja, hogy a képen fellelhető információt a pixelintenzitások által meghatározott térből egy ennél nagyobb absztrakciós szinten létező képjellemzők által meghatározott térbe transzformálja. Ezeket a képjellemzőket úgy tervezzük meg, hogy az általuk meghatározott térben könnyen elválaszthassuk a feladat szempontjából releváns információkat a zavaró hatásoktól. Az utolsó lépés egy döntési fázis, amelyben az algoritmus a képjellemzők alapján címkét rendel az adott képhez.

Az eddigiekben részleteztük a tanuló módszerek általános elméletetét, most azonban vizsgáljunk meg néhány konkrét alkalmazást is.

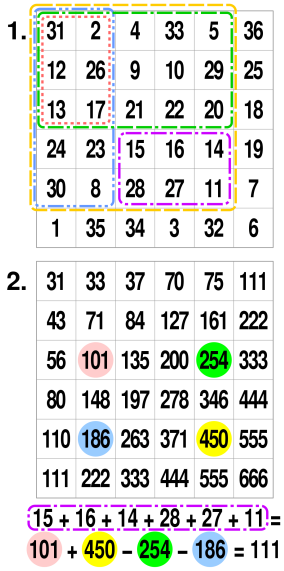
7.5.1. Viola-Jones

A Viola-Jones (más nevén Haar Cascade) detektor egy általános objektumfelismerő eljárás, amit azonban a legtöbb esetben arcdetektálásra szokás alkalmazni. Ez az eljárás egy speciális képjellemző fajtát használ fel, amit Haar-típusú jellemzőknek hívunk. A Haar jellemzők egy képrészletet egy bináris ablak segítségével vizsgálják úgy, hogy a fehér részek alá eső pixelek intenzitásainak összegéből kivonják a fekete részek alá eső pixelekét. Így egy előjeles számot kapunk, amely az ablak és a képrészlet hasonlóságát írja le. Nagy pozitív eredmény a hasonlóságot, a nagy negatív az ellentétességet, míg a nulla körüli eredmény a hasonlóság teljes hiányát jelenti.

A Haar jellemzők alkalmazásának egyik fontos hátránya, hogy egy aránylag kis méretű (mondjuk 24x24) képrészlet esetén is több, mint 160,000 különböző jellemzőt definiálhatunk, amelyek kiszámítása hatalmas számításigénnyel járna. Ennek csökkentésére számos módszert alkalmazunk,



7.9. ábra. *A Haar-féle jellemzők (bal), és ezek felhasználása arcdetektálásra (jobb).*



7.10. ábra. *Az integrálkép: Felül az eredeti kép, alul az ebből generált integrálkép értékei láthatók. A lilával keretezett terület pixeleinek összegét az alábbi módon számíthatjuk hatékonyan: a sárgával keretezett rész összegéből kivonjuk a zölddel, és a képkkel keretezett részek összegét. Ekkor azonban a pirossal keretezett rész összegét kétszer vontuk ki, így ezt hozzá kell adni. Így három összeadással bármilyen téglalapba eső pixelek összege számolható.*

melyek egyike az integrálkép kiszámítása, amely segítségével az egyes jellemzők válaszának számítása jelentős mértékben gyorsítható.

Ezzel együtt azonban még mindig rengeteg jellemzőnk van, amelyen nagy része valószínűleg nem is segít az adott objektum felismerésében. A hasznos jellemzők kiválasztására egy tanító adatbázist alkalmazunk, amelynek minden elemét az alábbi módon osztályozzuk:

$$label(I) = sign\Big(\sum_{j=1}^{M} \alpha_j h_j(I)\Big) \quad ahol \quad h_j(I) = \begin{cases} s_j & \text{ha } f_j \geq \theta_j \\ -s_j & \text{ha } f_j < \theta_j \end{cases} \tag{7.9}$$

Ahol α_j a j-edik jellemző relatív súlya, f_j a jellemző válasza, θ_j pedig a hozzá tartozó küszöbérték. A jellemzőkhöz tartozó küszöbértékeket, és s_j értékeket minden egyes jellemzőhöz külön-külön határozzuk meg úgy, hogy az egyetlen jellemző alapján elvégzett osztályozás minimális hibával történjen. Ezt követően az adott jellemző súlyát úgy választjuk meg, hogy az a hibás osztályozások számával fordítottan arányos legyen. Érdemes megjegyezni, hogy az egyes képek osztályozását nem egyenlő súllyal vesszük figyelembe: a már más jellemzők által helyesen osztályozott képek súlyát csökkentjük.

Ezzel a módszerrel a kis súlyú jellemzők elvetésével a kezdeti 160,000 jellemzőt nagyjából 6000-re lehet csökkenteni. Ez azonban még mindig rengeteg, hiszen a detekció során minden egyes pozícióban ezeket ki kell értékelni. Éppen ezért a detektor készítői a jellemzőket egy kaszkád architektúrában helyezik el: ennek első szintjén összesen két jellemző van, melyek az arcszerű képek 100%-át képesek detektálni, a nem arcszerű részletek 50%-át viszont el tudjuk vetni segítségükkel. Ezt követően a megmaradt képrészleteken lefuttatjuk a következő szűrőréteget, melyben 10 újabb szűrő található, amik a nem arcszerű képrészletek nagy részét képesek kiszűrni. Könnyen belátható, hogy ezzel a megoldással a negatív képrészletek túlnyomó többségét nagyon kevés számítás árán elvethetjük, így a teljes osztályozót csak néhány képrészleten kell lefuttatni.

7.5.2. Bag of (Visual) Words

A Viola-Jones detektor egyik hátránya, hogy a felhasznált jellemzők nem invariánsak a korábban megismert képi transzformációkra, így a segítségükkel elvégzett algoritmus is csak bizonyos mértékig képes ezt a problémát kompenzálni. Erre a problémára jó megoldást nyújthatnak a lokális képjellemzők, amelyek a sarokszerű pontok detektálása mellett egy számos transzformációra invariáns leírókódot is számítottak minden detektált jellemzőhöz. Ezek a jellemzők egyszerre kódolják a kép kinézetét és struktúráját egy kis lokális szeletben, relatív elhelyezkedésükből pedig a globális formára is következtetni lehet. Legnagyobb hátrányuk, hogy számításuk meglehetősen költséges: a korábban bemutatott SIFT-algoritmus futási ideje egy átlagos méretű képre asztali gépen is a másodperces nagyságrendben van.

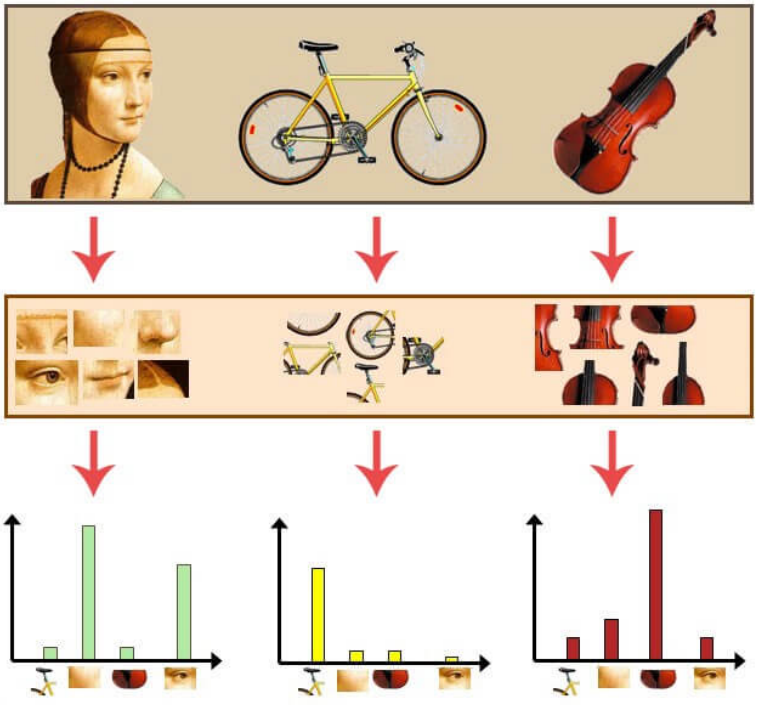
Lokális gradienshisztogramra épülő képjellemzők alapján történő osztályozásra leggyakrabban vizuális szóhalmazra (angolul: Bag of Visual Words) alapuló eljárásokat szoktak alkalmazni. A szóhalmazra alapuló osztályozás a szöveges dokumentumok osztályozásának tudományterületéről származik. Az eljárás alapötlete, hogy egy szöveget téma alapján be lehet sorolni a szövegben előforduló szavak relatív gyakorisága alapján. Fontos megjegyezni, hogy a módszer csupán a szavak előfordulását veszi figyelembe, azok sorrendjét, relatív helyzetét figyelmen kívül hagyja.

A módszer könnyedén átalakítható képek osztályozásának esetére, amennyiben a különböző lokális képjellemzőket vizuális szavakként értelmezzük. Az átalakítás során azonban egy problémába ütközünk: a szóhalmazos osztályozás során rendelkezésünkre állt egy szótár, amely alapján az egyes szavakat le tudtuk kódolni (például az alapján, hogy az adott szó hányadik helyen szerepel a szótárban). A szavak betűnkénti egyezése alapján a szótárban lévő szavakkal meg tudtuk őket feleltetni, valamint egy szinonimaszótár segítségével még az azonos jelentésű szavakat is képesek voltunk azonos kóddal szerepeltetni.

A vizuális szavak esetén azonban nem áll rendelkezésünkre ilyen szótár. A különböző vizuális szavak ráadásul valójában lebegőpontos számok sorozatai, amelyek sosem fognak egymással elemenként megegyezni. Így felmerülhet a kérdés: hogyan tudunk egy képi adatbázisból vizuális szótárt konstruálni, és hogyan tudjuk a képeken talált vizuális szavakat a szótárban szereplő szavak közé beosztani?

Erre a korábban szegmentálásra használt klaszterezés művelete a megoldás, amely a felügyelet nélküli gépi tanulás egyik alapvető módszere. A klaszterezés során a vizuális szóhalmazt úgy osztjuk kisebb csoportokra (klaszterekre), hogy a csoportok minél kompaktabbak legyenek, vagyis a csoport elemeinek a középtől számított távolságainak összege minimális legyen. A klaszterek számának a megválasztása alapvetően a tervező feladata, akinek egy kompromisszumértéket kell találnia a klaszterek kompaktságának csökkentése és a csoportszám túlzott növelése között.

Amennyiben a klaszterezés segítségével sikeresen konstruáltunk egy vizuális szótárt, ennek szavaihoz az új lokális képjellemzők már a szóközepektől számított négyzetes hiba minimalizálásával könnyedén beoszthatók. Ha ezt megtettük, az adott képen megtalálható vizuális szavakból egy hisztogramot konstruálhatunk, amely azt fejezi ki, hogy a vizuális szótárban található szavak közül melyik milyen relatív gyakorisággal fordul elő a képen. Ezt a hisztogramot súlyozhatjuk úgy, hogy az egyes szavakat nem keményen rendeljük az egyes szótárbeli szavakhoz, hanem az alapján súlyozva, hogy azok a középponttól milyen távolságra voltak. Ezen a módon a nagy konfidenciával észlelt szavak nagyobb súllyal fognak szerepelni.



7.11. ábra. *A vizuális szavak modellje.*

Ezt követően a hisztogramot felhasználhatjuk úgy, hogy a képeket bizonyos osztályokhoz rendeljük. A döntés elvégzéséhez számos módszert használhatunk, amelyek általában a gépi tanulás tárházából kerülnek ki. A gépi tanuló algoritmusok általánosságban egy adatbázisból nyerik ki az optimális döntésfüggvényt valamilyen statisztikai vagy optimalizáló módszer segítségével. Az egyik legegyszerűbb példa a tanuló algoritmusra a kNN-módszer, amely ebben az esetben könnyedén alkalmazható. A döntéshez vesszük azt a képi adatbázist, amelyik segítségével a szótárt konstruáltuk (innentől: tanító adatbázis), majd ebben megkeressük a jelenlegi kép k legközelebbi szomszédját, ezúttal azonban nem a pixelértékek, hanem a szóhalmazhisztogram alapján számítunk távolságot. Az új kép címkéjét pedig a k legközelebbi szomszéd dönti el többségi szavazás alapján. Fontos megjegyezni, hogy a gépi tanulás megoldásaihoz általában ismernünk kell az adatbázis címkéit.

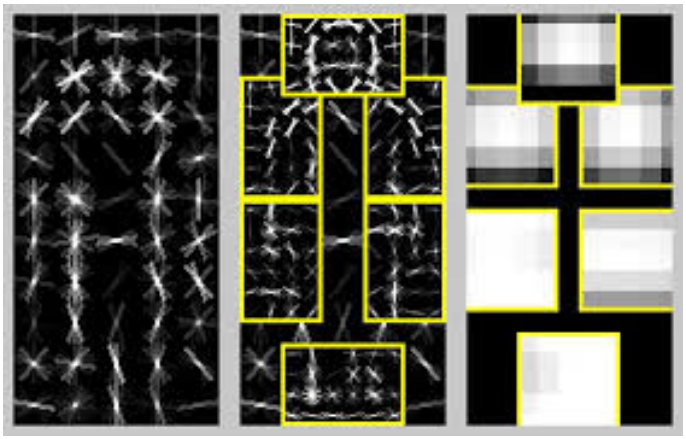
7.5.3. Deformálható rész-modellek

Felmerülhet azonban a kérdés, hogy mi történik, ha az egyszerű osztályozáson túlmenően az objektum pozícióját is meg szeretnénk határozni (vagyis detektálni szeretnénk). Lehetséges-e ilyen esetben a vizuális szóhalmazos módszert alkalmazni? A válasz, hogy alapvetően ugyan lehetséges, a kinyert pozícióinformációk azonban rendkívül pontatlanok lesznek, ugyanis a szóhalmazmódszer semmilyen az egyes vizuális szavak abszolút vagy relatív helyzetére vonatkozó információt nem használt fel az osztályozás elvégzéséhez.

Ez a hiányosság azonban egyszerűen orvosolható néhány extra komponens bevezetésével, amit a deformálható részmodellek meg is tesznek. E módszerek lényege, hogy az egyes osztályokat nem egy vizuális szavakat tartalmazó halmazként, hanem szavakból álló gráfként írják le, ahol a gráf élei az egyes szavak közti geometriai kapcsolatokat írják le. Ezek a kapcsolatok nem teljesen merevek, hanem bizonyos korlátok között változhatnak (deformáció).

A deformálható részmodellek által felhasznált jellemzők alapvetően konvolúciós szűrők, melyeknek két fajtája van: a központi (root) és a rész (part) szűrők. Az előbbi feladata, hogy az objektum közepénél adjon nagy választ, míg az utóbbiaknak az korábban említett objektumrészek pozíciójában kell nagy választ adniuk. A modell további szabad paraméterei az egyes rész szűrők a központihoz viszonyított relatív pozíciói, amely minden rész szűrő esetén két extra paramétert jelent. Ezeket a szabad paramétereket az algoritmus tanító adatok alapján tanulja meg.

7.12. ábra. *A deformálható részmodell.*



7.13. ábra. *A központi szűrő (bal), a rész szűrők (közép) és a deformációk (jobb).*

Akad azonban egy probléma: az algoritmus tanításánál ugyanis csak az objektum pozíciójáról van információnk, az egyes részek helyzete nincs kikötve. Éppen ezért nem világos, hogy a rész szűrőket pontosan mire is tanítsuk, hiszen nem tudjuk, hogy pontosan hol kellene maximális választ adniuk. Más szóval a szűrők elvárt maximum pozíciója, valamint a központi szűrőtől számított relatív helyzetük ún. látens paraméterek.

Ennek feloldására a következő iteratív eljárást alkalmazzuk:

1. A szűrő paraméterek és a relatív elmozdulások véletlen inicializálása.
2. Konvergenciáig:
   (a) A rész szűrőket lefuttatjuk a pozitív detekciók körül, és a válaszuk maximumát előírjuk, mint elvárt helyes választ.
   (b) A modell paramétereinek tanítása az így egyszerűsített példán.

Érdemes megjegyezni, hogy az egyes rész szűrők a véletlen inicializálás miatt valószínűsíthetően különböző részek felismerésére fognak konvergálni. Észrevehetjük, hogy a tanítás két lépése analóg a korábban megismert k-Means és Expectation-Maximization algoritmushoz: nem véletlen, hiszen a

klaszterezés esetén is látens változókkal kellett dolgoznunk, csak még nem ismertük ezt a problémát ezen a néven.

Fontos megjegyezni, hogy ahogyan az eredeti vizuális szóhalmazeljárás igényelte, hogy a szótár konstruálásához használt adatbázisban a képek legyenek felcímkézve az azon megtalálható osztályok szerint, a deformálható részmodelleknek ezen felül szüksége van az objektumok pozícióinformációjára is. Ilyen adatbázisokat első sorban kézzel szoktunk előállítani, majd ezután az így elkészült algoritmus képes lesz arra, hogy más, eddig ismeretlen képekre is az adott műveletet elvégezze.

További Olvasnivaló

[1] R. Szeliski, *Computer Vision*. Springer London, 2011. DOI: 10.1007/978-1-84882-935-0. cím: https://doi.org/10.1007%2F978-1-84882-935-0.

[2] L. Ross, "The Image Processing Handbook, Sixth Edition, John C. Russ. CRC Press, Boca Raton FL, 2011, 972 pages. ISBN 1-4398-4045-0(Hardcover)", *Microscopy and Microanalysis*, 17. évf., 5. sz., 843–843. old., 2011. szept. DOI: 10.1017/s1431927611012050. cím: https://doi.org/10.1017%2Fs1431927611012050.

[9] P. Viola és M. Jones, "Rapid object detection using a boosted cascade of simple features", *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, IEEE Comput. Soc. DOI: 10.1109/cvpr.2001.990517. cím: https://doi.org/10.1109%2Fcvpr.2001.990517.

[10] P. F. Felzenszwalb, R. B. Girshick, D. McAllester és D. Ramanan, "Object Detection with Discriminatively Trained Part-Based Models", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32. évf., 9. sz., 1627–1645. old., 2010. szept. DOI: 10.1109/tpami.2009.167. cím: https://doi.org/10.1109%2Ftpami.2009.167.

[11] R. O. Duda és P. E. Hart, "Use of the Hough transformation to detect lines and curves in pictures", *Communications of the ACM*, 15. évf., 1. sz., 11–15. old., 1972. jan. DOI: 10.1145/361237.361242. cím: https://doi.org/10.1145%2F361237.361242.

[12] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm", *IEEE Transactions on Information Theory*, 13. évf., 2. sz., 260–269. old., 1967. ápr. DOI: 10.1109/tit.1967.1054010. cím: https://doi.org/10.1109%2Ftit.1967.1054010.

[13] H. W. Kuhn, "The Hungarian Method for the Assignment Problem", *50 Years of Integer Programming 1958-2008*, Springer Berlin Heidelberg, 2009. nov., 29–47. old. DOI: 10.1007/978-3-540-68279-0_2. cím: https://doi.org/10.1007%2F978-3-540-68279-0_2.

II. rész

Tanuló Látás

8. fejezet

Neurális hálózatok

Az előző részben ismertettük a hagyományos látórendszerek lépéseit, melynek utolsó szakaszában egy rövid ízelítőt adtunk a gépi tanulás módszereiből is. A modern, mesterséges intelligenciára alapuló látórendszerek valójában nem abban különböznek a hagyományos módszerektől, hogy ezek használják a gépi tanulást (hiszen az utolsó lépésben a hagyományos módszerek nagy része is ezeket alkalmazza), hanem abban, hogy a tanuló látórendszerek szinte kizárólag csak tanuló elemekből épülnek, vagyis úgynevezett end-to-end tanulást valósítanak meg.

A következő fejezetekben ezen módszerek elméletét fogjuk ismertetni, kezdve a neurális hálós rendszerekben alkalmazott tanuló modellel. Ezt követően megismerjük az alkalmazott költségfüggvényeket, és azok optimalizálására alkalmazott módszereket. Az ezt követő fejezetekben megismerkedhetünk a képi információk feldolgozására kifejlesztett konvolúciós architektúrákkal, majd pedig a neurális hálózatok taníásának gyakorlati praktikáiba nyerhetünk betekintést. Végezetül kitárgyaljuk a magasabb szintű feladatok (szegmentálás, detektálás, dinamika) megoldását.

8.0.1. A Perceptron modell

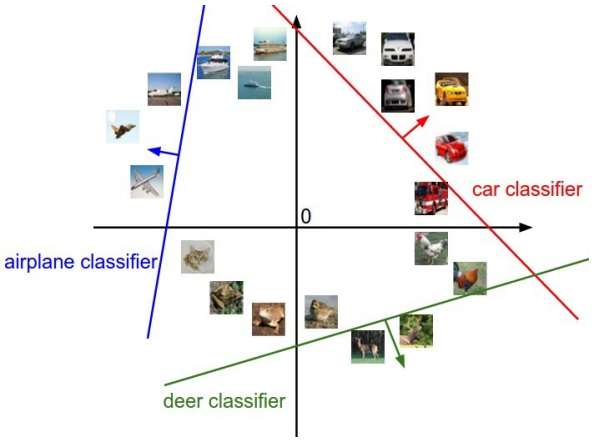
A mély tanuló rendszerek alapeleme egy lineáris osztályozó algoritmus, amelynek számos elnevezése létezik. Gyakran szokás perceptron, illetve neuron elnevezéssel illetni, valamint – osztályozó jellege ellenére – logisztikus regresszió néven is ismert. Az algoritmus működésének lényege, hogy a kép pixeleit egyetlen vektorba rendezi, majd ezt a vektort egy súlymátrixszal szorozza meg, így egy kimeneti vektort állítva elő, aminek annyi eleme van, ahány osztály között döntenünk kell. Ennek a vektornak minden egyes eleme értelmezhető úgy, mint az egyik osztály „jósági" értéke, vagyis minél nagyobb, annál inkább tartozik a kép az adott osztályba. Formálisan a következőképp adható meg a perceptron modellje:

$$s = Wx \tag{8.1}$$

Ahol x a bemenet, s az osztály jóság, W pedig a paraméterek, vagy súlyok mátrixa (ezt a jelölésrendszert a jelen fejezetben következetesen alkalmazzuk). Az osztályozás ilyen módját úgy lehet elképzelni, hogy a W mátrix i-edik sora kijelöl egy olyan irányt a pixelek terében, amerre az i-edik osztály jósága nő. Ennek alapján az egyes osztályok közötti döntési határok egyenes szakaszokból tevődnek össze (bináris esetben egyetlen egyenes/hipersík).

8.1. A tanítás módszere

Miután definiáltuk a Perceptron algoritmus modelljét és elemeztük annak működését, itt az ideje, hogy a modell helyes működéséhez szükséges W súlymátrix elemeinek meghatározásáról is szót ejtsünk. A módszer alapvető kérdése ugyanis, hogy hogyan lehet a súlyok értékét úgy meghatározni, hogy az osztályozás minél pontosabb legyen, valamint, hogy milyen költségfüggvény segítségével mérhető jól az algoritmus teljesítménye.



8.1. ábra. *A lineáris osztályozás esetében az egyes osztály jósági értékek a tér egy irányában lineárisan nőnek, míg a többiben konstansak. A növekedés irányát a súlymátrix adott osztályhoz tartozó sorverktora határozza meg.*

8.1.1. Hibafüggvények

Első nekifutásra célszerű lehet az osztályozás minőségét a jól eltalált tanító adatok arányával jellemezni, ez azonban nem képes különbséget tenni egy azonos pontosságú, de eltérő bizonytalanságú osztályozás végző modell között. Éppen ezért a modell kimenete és az adott tanítóadathoz előírt kimenet között egészen új költségfüggvényeket fogunk definiálni, melyeknek az egész tanító adathalmazra vett átlaga megadja a teljes hiba mértékét. Célszerűnek tűnhet egyszerűen az elvárt és a becsült kimenet közti négyzetes hibát venni, amely regressziós problémák esetén a leggyakrabban használt hibafüggvény. A kimeneti érték numerikus közelítése viszont osztályozás esetében nem feltétlenül praktikus, és habár a négyzetes hiba ilyen esetekben is használható, mégis könnyedén lehet jobb hibafüggvényeket konstruálni.

Az egyik gyakran használt hiba az úgynevezett Hinge, vagy SVM hibafüggvény. Ennek a hibafüggvénynek az alapelve, hogy definiálunk egy mennyiséget, amit résnek nevezünk, és ha a helyes osztály jósága legalább ezzel az értékkel nagyobb az összes többi jóságnál, akkor a hiba értéke 0. Ellenkező esetben a hiba értéke lineárisan nő. Ez a hibafüggvény felfogható egyfajta "biztonságos" elválasztást előíró kritériumként. Az SVM hiba formálisan a következő:

$$Li = \sum_{j \neq y_i} max(0, s_j - s_{corr} + \Delta) \tag{8.2}$$

Ahol s_j a j-edik, s_{corr} pedig a helyes osztály jóság értéke.

8.2. ábra. *A Hinge költség szemléletesen.*

Létezik egy másik gyakorlatban elterjedt hibafüggvény, amelyik a geometriai szemlélet helyett inkább a valószínűségszámítás oldaláról közelíti meg a problémát. Ez a költségfüggvény az entrópia fogalmát használja fel. Az entrópia fogalma arra épül, hogy ha különböző valószínűséggel történő eseményeket szeretnénk elkódolni, akkor nem érdemes minden eseményre ugyanannyi bitet szánni, a valószínű eseményeket kevés, míg a valószínűtleneket sok biten érdemes ábrázolni, így az összes esemény közlésére elhasznált bitek mennyiségét minimalizálni lehet. Egy p valószínűséggel bekövetkező eseményt a p logaritmusának reciprokával megegyező számú biten érdemes kódolni. Ezt felhasználva az entrópia megadja az összes eseményre elhasznált bitek számának várható értékét:

$$H(p) = -\sum_i p_i log p_i \tag{8.3}$$

Belátható azonban, hogy ha a p valószínűségi eloszlást nem ismerjük, hanem csak egy közelítő q eloszlást, akkor az optimálisnál csak nagyobb eredményt kapunk. Ezt a nagyobb értéket fejezi ki a keresztentrópia mértéke. Persze minél inkább közelíti a q eloszlás a p-t, annál inkább csökken a keresztentrópia. A két entrópiafajta különbségét KL divergenciának nevezzük, ami egy szigorúan nemnegatív függvény, amelyet gyakran használnak valószínűségi eloszlások hasonlósági mércéjének.

$$H(p, q) = -\sum_i p_i log q_i \tag{8.4}$$

A keresztentrópia felhasználható osztályozási hibafüggvényként az alábbi módon: első lépésként a modell kimeneti jóságait egy SoftMax nevű normalizáló függvény segítségével valószínűség jellegű értékekké konvertáljuk. Ez a függvény minden értéket a $[0, 1]$ tartományba transzformál úgy, hogy az értékek összege pontosa egy legyen. A SoftMax függvény az alábbi módon írható fel:

$$q_{k,i} = q(y_k|x_i) = \frac{e_{s_k}}{\sum_j e_{s_j}} \tag{8.5}$$

Innen a hibafüggvényt úgy definiáljuk, mint a címkék elvárt eloszlása, és a becsült q eloszlás közti keresztentrópia. Mivel a keresztentrópia akkor minimális, ha a két eloszlás megegyezik, ezért ennek a függvénynek a minimalizálásával a becsült valószínűségek az előírtakhoz fognak tartani. A címkék előírt eloszlását úgy konstruáljuk, hogy a helyes osztály elvárt valószínűségét 1-nek, míg az összes többiét nullának választjuk. Így a keresztentrópia hibafüggvény az alábbi alakra egyszerűsödik:

$$\begin{aligned} L_i = H(p_i, q_i) &= -\sum_k p_{k,i} log q_{k,i} \\ L_i &= -log q_{true,i} \end{aligned} \tag{8.6}$$

Ahol $p_{k,i}$ és $q_{k,i}$ az i-edik tanító adat k-adik osztályhoz tartozó előírt és becsült valószínűségei, $q_{corr,i}$ pedig a helyes osztály becsült valószínűsége. A keresztentrópia költségfüggvény egyik előnye, hogy nehezen értelmezhető „jóság” értékek helyett valószínűség jellegű értékekkel dolgozik, így a modell kimenete könnyebben felhasználható. Hátránya az SVM hibával szemben, hogy a költség értéke sosem lesz nulla, vagyis az SVM hiba „takarékosabb”: megelégszik a biztonságos elválasztással, és hagyja, hogy a modell a megmaradt erőforrásait a többi tanító adat helyes osztályozására fordítsa. A gyakorlatban a két költségfüggvény közti különbség azonban alig kimutatható.

8.1.2. Regularizáció

Mindkét hibafüggvénynek van azonban egy alapvető problémája. Könnyű ugyanis belátni, hogy mindkét költségfüggvény esetén, ha egyszerűen a modell aktuális súlymátrixát egy nagy számmal megszorozzuk, akkor a hibafüggvények értéke csökkenni fog, az osztályozás pontossága viszont változatlan marad, hiszen egyszerűen minden kimeneti jóság ugyanazzal a konstanssal szorzódik. Ennek következtében a súlymátrix normája minden határon túl növekedni fog, ami egyrészt numerikus problémákhoz, másrészt egy túl magabiztos modellhez fog vezetni.

Éppen ezért ezeket a hibafüggvényeket nem önállóan, hanem egy regularizációs büntetőtaggal együtt szoktuk használni, ami a súlymátrix normáját tartja kordában. Elterjedt megoldás a mátrixnak az L1, illetve az L2 normáját használni büntetőtagként. Létezik ezen felül még az úgynevezett elasztikus regularizáció, amikor a kétfajta norma súlyozott átlagát használják. A végső hibafüggvény a következőképp adódik:

$$
\begin{aligned}
L &= \sum_{i}^{N} L_i + \lambda R(W) \\
R_{L1}(W) &= \sum_{k} \sum_{l} |W_{k,l}| \\
R_{L2}(W) &= \sum_{k} \sum_{l} W_{k,l}^2 \\
R_{EL}(W) &= \sum_{k} \sum_{l} \left(\beta W_{k,l}^2 + |W_{k,l}|\right)
\end{aligned}
\tag{8.7}
$$

Ahol L_i az i-edik tanítóadatra számolt hiba, N a tanító adatok száma, R a regularizációs tag, λ pedig a regularizáció relatív súlyát befolyásoló hiperparaméter. Érdemes megjegyezni, hogy a következő alfejezetben tárgyalt többrétegű hálók esetén a súlymátrix normájának növekedése túlillesztést okozhat, így ennek az elkerülése regularizáció.

8.2. Optimalizáció

Az előző előadás egyik legnagyobb lezáratlan kérdése, hogy miként lehet a korábban említett perceptron modell hibafüggvényét minimalizálni. Ebből kifolyólag az első témánk a hibafüggvények minimalizálására szolgáló optimalizálási módszerek tárgyalása. A perceptron súlyainak optimális értékére nem létezik zárt alakú megoldás, így iteratív optimalizálásra lesz szükség.

8.2.1. Gradiens-alapú optimalizálás

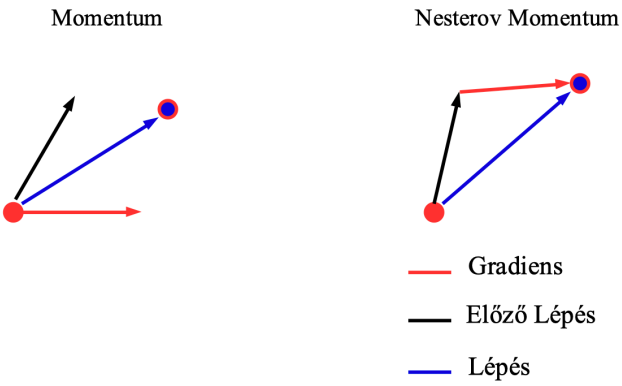
Szerencsére azonban a modell és a költségfüggvény is deriválható, így alkalmazhatunk gradiens alapú módszereket. Ha kiszámoljuk a hibafüggvény súlyok szerinti deriváltját (más szóval a súlyok gradiensét), akkor megkapjuk azt, hogy hogyan kellene a súlyoknak megváltozni ahhoz, hogy a hibafüggvény a lehető leggyorsabban növekedjen. Ha azonban a súlyokat a gradienssel ellenkező irányba változtatjuk, az a leggyorsabb csökkenés iránya lesz. Ezt a lépést egyfajta „fordított hegymászóként" ismételve egy idő után lokális minimumba jutunk.

Vegyük azonban észre, hogy mivel a teljes hibát szeretnénk minimalizálni, ezért minden egyes lépéskor ki kell értékelni az egész tanító adatbázis hibáját. Mivel a gradiens módszer aránylag sok lépés után konvergál csak, ezért ez nem praktikus. Éppen ezért a tanító adatbázist egyenlő méretű, véletlenszerűen kiválasztott részhalmazokra (minibatch-ekre) osztjuk, és minden egyes minibatch után végzünk egy lépést. Ezt a módszert sztochasztikus gradiens módszernek (SGD - Stochastic Gradient Descent) nevezzük. A minibatchek mérete általában a kettő hatványa szokott lenni, implementációs okokból. Érezhető persze, hogy az egyetlen minibatch-re számolt gradiens nem egyezik teljesen meg az egész adatbázisra számolttal, de elég közel van ahhoz, hogy megközelítőleg a jó irányba haladjon a módszer. Mivel így egyetlen lépést sokkal olcsóbb végrehajtani, összességében jelentős gyorsulást érünk el. A gradiens módszer képlete a következő:

$$
W_{k+1} = W_k - \alpha \frac{\partial ||E||^2}{\partial W} \tag{8.8}
$$

Ahol α a tanulási ráta, ami egy olyan hiperparaméter, ami nagymértékben befolyásolja a tanítás sebességét és minőségét. Helyes megválasztásáról egy későbbi fejezetben beszélünk részletesen.

Fontos még észrevenni, hogy a gradiens módszer egyik hátránya, hogy könnyen beragadhat lokális minimumokba. Ennek megoldására az inverz hegymászó ötletét le kell cserélnünk a hegyről leguruló szikla képére. Más szóval élve a gradiens módszerhez egyfajta tehetetlenséget, momentumot veszünk hozzá. Ezt a gyakorlatban úgy tesszük, hogy az adott időpontban elvégzett lépés a negatív gradiens iránya és az eggyel korábbi időpillanatban tett lépés súlyozott átlagából tevődik össze. Ez a súly alkalmazástól függően általában a [0,1-0,9] tartományban mozog.



8.3. ábra. *A momentum módszer (bal) és a Nesterov-momentum (jobb). Ez utóbbi előbb lép a momentum irányba, majd ott számolja ki a gradienst, ami valamivel stabilabb működést eredményez.*

A sztochasztikus gradiens módszer egyik hiányossága, hogy a sokdimenziós paramétertér minden irányát egyenértékűnek veszi. Előfordulhat azonban, hogy a költségfüggvény az egyik irányban meglehetősen meredek, így ebbe az irányba óvatoson érdemes haladni, mivel egy nagyobb ugrással könnyen átugorhatjuk a minimum helyét. Eközben egy másik irányban a költségfüggvény lapos, az optimum pedig meglehetősen messze található. Míg az első probléma a lépésméret csökkentését igényelné, a második esetben pont növelni kellene, a kettőt egyszerre pedig nem lehet.

Erre a problémára ad megoldást az AdaGrad és az RMSProp módszer. Mindkettő alapelve az, hogy a gradiensek nagyságát megjegyzik az egyes irányokban, a lépésméretet pedig ezek inverzével skálázzák, így biztosítva az eltérő lépés méretet az egyes irányokban. Az optimalizáló algoritmusok közül az egyik leginkább elterjedt az Adam algoritmus, amely a momentum és a gradiens skálázás módszerét kombinálja.

8.2.2. Magasabb deriváltak

A gradiens módszernek van azonban két meglehetősen nagy hátránya: egyrészt a fix lépésméret következtében az optimum közelében a módszer oszcillálni kezd, hiszen nem tud pontosan az optimum pozícióba lépni. Ezen felül a módszer legtöbbször meglehetősen lassú, különösképp, ha az optimumtól messze lévő pontból indul.

Felmerülhet azonban bennünk, hogy amennyiben a minimalizálandó költségfüggvényt nem egy elsőhanem egy másodrendű függvénnyel közelítenénk, akkor ennek a minimumpontját analitikusan kiszámíthatjuk, és egyből bele is ugorhatunk. Ezen az elven működik a Newton-módszer, amely a fentiek alapján a függvényt az adott x_N pontban a másodfokú Taylor-polinomjával közelíti az alábbi módon:

$$f(W_k + \Delta W) \approx f(W_k) + f'(W_k)\Delta W + \frac{1}{2} f''(W_k)\Delta W^2 \tag{8.9}$$

Amit ΔW szerint deriválva és azt nullával egyenlővé téve megkaphatjuk a minimum helyet:

$$\begin{aligned} 0 &= f'(W_k) + f''(W_k)\Delta W \\ \Delta W &= -\frac{f'(W_k)}{f''(W_k)} \\ W_{k+1} &= W_k - \frac{f'(W_k)}{f''(W_k)} \end{aligned} \tag{8.10}$$

Természetesen, mivel a függvény valójában nem másodfokú, ezért a fenti képletet iterációs szabályként alkalmazzuk. Természetesen a Newton módszer többváltozós függvények esetében hasonló alakban működik, ekkor a függvény első deriváltja helyett a gradienst, a második deriváltja helyett pedig az ún. Hesse-mátrixot (a másodrendű parciális deriváltakat tartalmazó mátrix) használhatjuk:

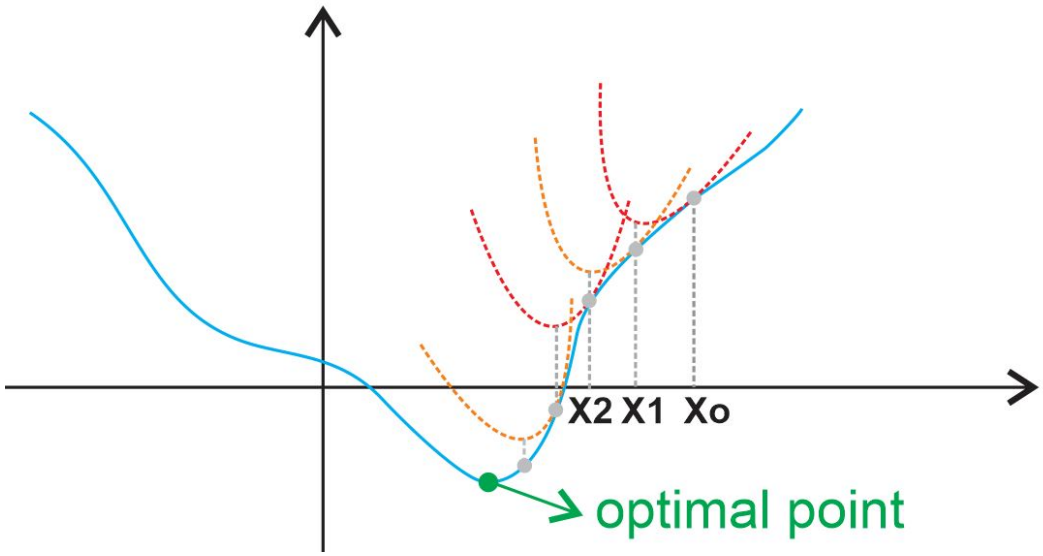
$$W_{k+1} = W_k - H_W^{-1} \quad \nabla_W f(W_k) \tag{8.11}$$

A módszernek azonban több problémája is van: egyrészt a Hesse-mátrix számolása gyakorta nehéz, ráadásul a mérete a változók számával négyzetesen nő, így egy több millió paraméterrel rendelkező gépi tanuló algoritmus esetében ez óriási lehet. Ezen felül további problémák adódhatnak, ha a Hesse-mátrix nem invertálható (vagyis valamelyik sajátértéke közel nulla). Ez abban az esetben fordulhat elő, ha a függvény valamelyik irányban lapos, vagy - sokkal valószínűbb - éppen nyeregpontban vagyunk.

Ennek a problémának az elkerülésére használható a Levenberg-Marquardt algoritmus. Ez az eljárás az alábbi lépésszabályt alkalmazza:

$$W_{k+1} = W_k - [H_W + \lambda I]^{-1} \quad \nabla_W f(W_k) \tag{8.12}$$

Azzal, hogy a Hesse-mátrixhoz az egységmátrix konstansszorosát hozzáadjuk biztosítjuk, hogy az invertálandó mátrix mindig pozitív definit legyen. Szemléletesen ebből az adódik, hogy ha a Hesse-mátrix "kicsi", akkor az egységmátrix inverze dominál, és a módszer a gradiens-módszerré egyszerűsödik, míg ha a Hesse-mátrix "nagy", akkor a Newton-módszert használjuk. Ennek hatására a problémás területeken az algoritmus - ha lassan is - de biztosan áthaladhat.

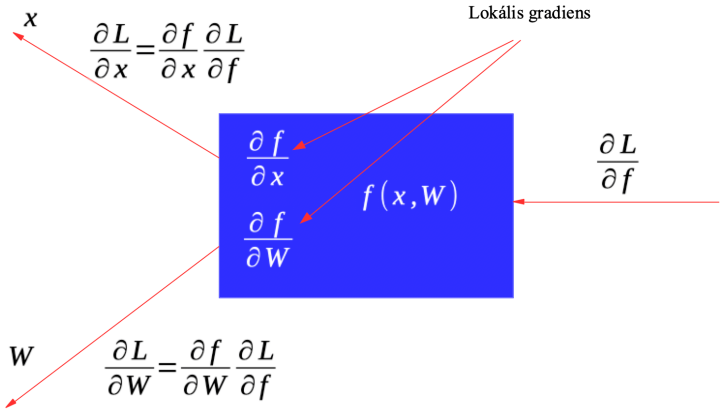


8.4. ábra. *A Newton algoritmus elve.*

8.2.3. Backpropagation

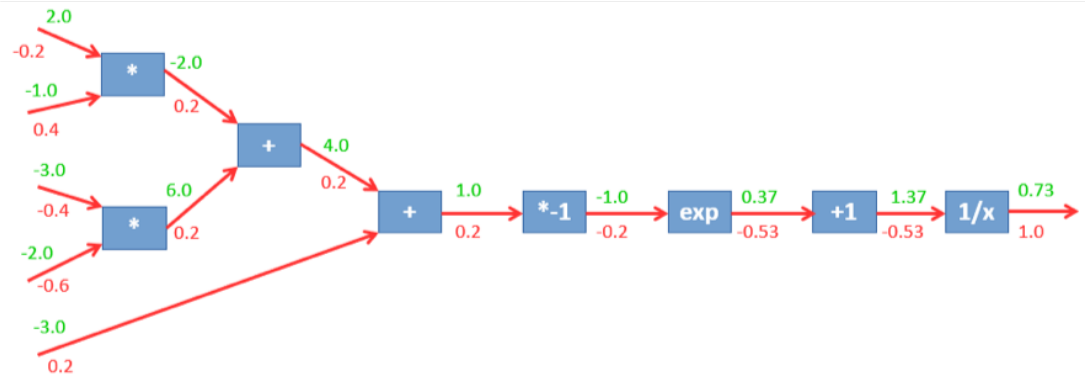
Amint azt már korábban említettük, az egyszerű lineáris modellek képességei erősen limitáltak, így a képi bemenetek esetében ritkán használjuk őket. Ismertetésük azonban szükséges volt, ugyanis ezek az egyszerű lineáris modellek könnyedén összeépíthetők komplex nemlineáris tanuló eljárásokká. Amennyiben az előző alfejezetben ismertetett neuron modelleket egymás után csatoljuk, akkor egy többrétegű, előrecsatolt neurális hálózatot kapunk. A neurális hálózatoknak minden rétegéhez tartozik egy ismeretlen súlymátrix, amelyet a korábban ismertetett költségfüggvények és optimalizálási módszerek segítségével határozhatunk meg.

Az egyetlen kérdéses lépés az a hibafüggvény súlyok szerinti deriváltjának számítása. Egy neurális háló elképzelhető, mint egy számítási gráf, ahol a gráf egyes csomópontjai egyszerű, analitikusan deriválható függvényeket implementálnak. Amennyiben egy számítási gráfban ismerjük a bemeneteket, és az egyes csomópontok által implementált függvényeket, akkor az összes csomópont kimenetét ki tudjuk számítani. Ezt nevezzük az előreterjesztés műveletének. Érdemes azonban észrevenni, hogy amennyiben ismerjük a csomópontok függvényeit, és ismerjük a számunkra érdekes mennyiség (ez esetben a hibafüggvény) deriváltját a csomópont kimenete szerint, akkor a deriválás láncszabályának segítségével könnyedén előállíthatjuk a csomópont bemenete és súlyai szerinti deriváltakat.



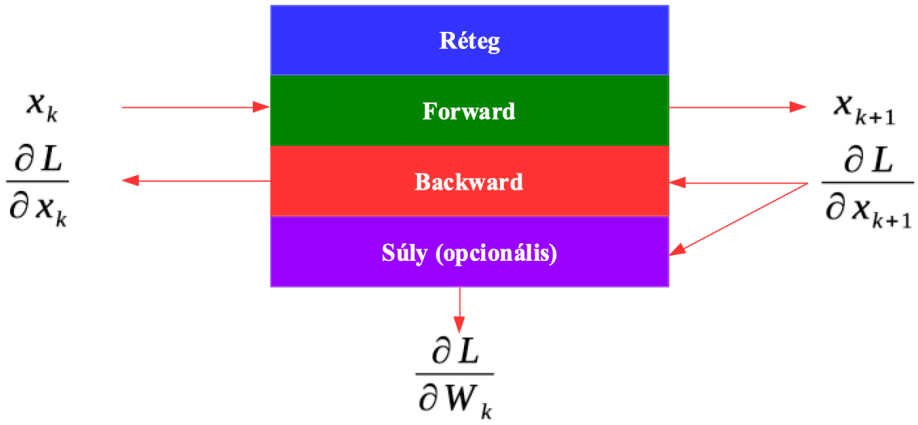
8.5. ábra. *A láncszabály szemléltetése.*

Ahol L a hibafüggvény, f és x az adott réteg ki- és bemenete, W pedig a réteg paraméterei. Ezzel a módszerrel a hálón hátrafele haladva az összes bemenet és súly gradiensét meg tudjuk határozni. Ezt a műveletet hátraterjesztésnek (backpropagation) nevezzük. Az egyetlen kérdés csupán, hogy hogyan kapjuk meg a hibafüggvény deriváltját a számítási gráf utolsó csomópontjának kimenete szerint. Vegyük észre azonban, hogy az előreterjesztés során legutolsó elvégzendő művelet pont a hibafüggvény kiszámítása, azaz a gráf utolsó pontjának a kimenete maga a hibafüggvény. Egy mennyiség saját maga szerinti deriváltja pedig triviálisan 1. Ily módon tehát minden rendelkezésre áll a háló gradienseinek számolására.



8.6. ábra. *A backpropagation végrehajtása egy példa gráfon. Zölddel az aktivációk, pirossal a deriváltak szerepelnek.*

Érdemes észrevenni, hogy a neurális háló elemeinek nem feltétlenül kell a korábbi fejezetben megismert perceptron függvényeknek lenniük. A valóságban a neurális hálózatok számos különböző fajta rétegből állnak, melyeknek közös tulajdonságuk, hogy deriválható függvényeket valósítanak meg. Saját magunk is könnyedén készíthetünk új típusú rétegeket, egészen addig, amíg az előre- és hátraterjesztés részfeladatait elvégző függvényeket megvalósítjuk.



8.7. ábra. *Egy réteg által biztosított, általános interfész, ami tartalmazza a tanításhoz és a predikcióhoz szükséges komponenseket is.*

További Olvasnivaló

[18] J. Heaton, "Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning", *Genetic Programming and Evolvable Machines*, 19. évf., 1-2. sz., 305–307. old., 2017. okt. DOI: 10.1007/s10710-017-9314-z. cím: https://doi.org/10.1007%2Fs10710-017-9314-z.

[19] G. James, D. Witten, T. Hastie és R. Tibshirani, *An Introduction to Statistical Learning*. Springer New York, 2013. DOI: 10.1007/978-1-4614-7138-7. cím: https://doi.org/10.1007%2F978-1-4614-7138-7.

[20] D. W. Marquardt, "An Algorithm for Least-Squares Estimation of Nonlinear Parameters", *Journal of the Society for Industrial and Applied Mathematics*, 11. évf., 2. sz., 431–441. old., 1963. jún. DOI: 10.1137/0111030. cím: https://doi.org/10.1137%2F0111030.

9. fejezet

Konvolúciós Neurális Hálók

9.1. Bevezetés

A korábbi előadásban megismerkedtünk a gépi tanulás alapjaival, ezen belül is a lineáris osztályozás módszerével. Azt is beláttuk azonban, hogy ezek az algoritmusok nem elég komplexek ahhoz, hogy képek osztályozását jó minőségben elvégezzék. Ennek ellenére szerencsére ezek az egyszerű osztályozók felhasználhatók, mint egy nagyobb mesterséges intelligencia modell építőkövei. A mostani előadás témája az egyszerű lineáris modellekből épített mély neurális hálók lesznek.

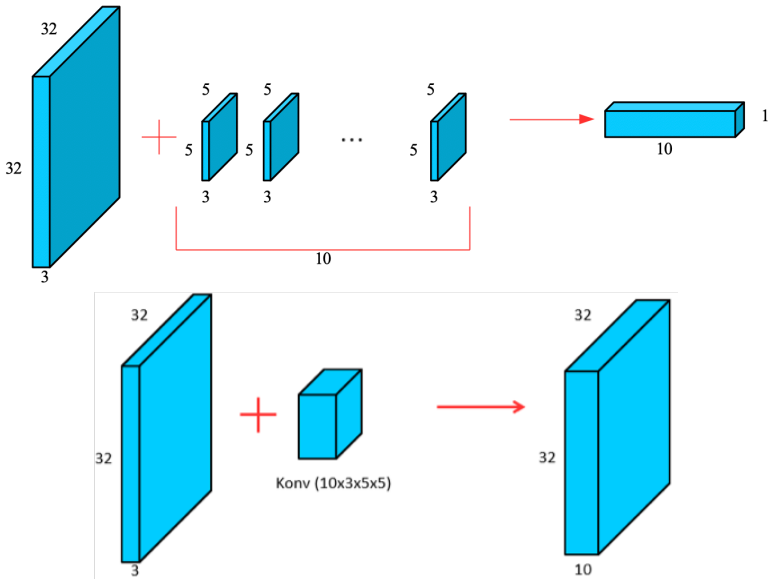
9.2. Konvolúciós neurális hálók

A korábban ismertetett lineáris neuron modell a számítógépes látásban használt hálókban ugyan előfordul, de tipikusan nem ilyen rétegekből épül fel a háló nagy része. Ennek oka, hogy a lineáris (más néven teljesen kapcsolt – fully connected) réteg minden bemenete és kimenete között létesít egy kapcsolatot, aminek következtében rengeteg paraméterrel rendelkezik, ami elősegíti a túlillesztés előfordulását, ráadásul a háló tárolását is megnehezíti. További hátránya, hogy a képek térbeliségét egyáltalán nem használja ki.

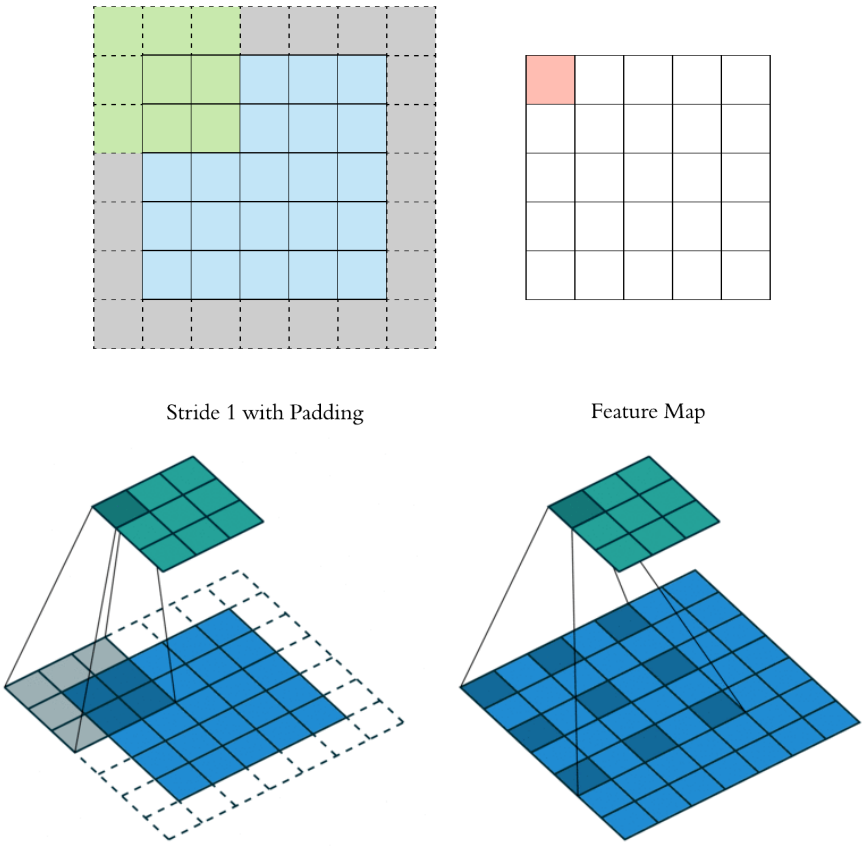
9.2.1. Konvolúciós réteg

Ezekre a problémákra ad megoldást a konvolúciós réteg, amely nevéből adódóan a korábbi kötetben ismertetett konvolúciós szűrőkhöz hasonlóan működik. Egy konvolúciós réteg a bemeneti (általában 1-3 csatornás) képen N darab különböző konvolúciós szűrőt futtat végig, amelynek eredménye egy N csatornás szűrt kép. Az ezt követő konvolúciós rétegek már N csatornás bemeneti képpel dolgoznak. A használt szűrők mérete és a csatornák száma (más néven a réteg mélysége) tipikus hiperparaméterek. Érdemes megjegyezni, hogy a gyakorlatban a legtöbb esetben a kép térbeli méretének megőrzése érdekében a kép széleit 0-kal egészítjük ki.

A konvolúciós rétegeknek még két fontos paramétere létezik: a stride, és a dilatáció. A konvolúciós szűrő egyes stride esetén minden pixelen végighalad, míg kettes stride esetén minden másodikon, és így tovább. Ezt a beállítást a kép térbeli méretének csökkentésére szokták használni. A dilatáció (9.2. ábra, jobb alul) értéke azt határozza meg, hogy a szűrőn eggyel arrébb található súlyt a képen hánnyal arrébb lévő pixellel szorozzuk. Egyes dilatáció értéke esetén a szűrő a megszokott módon viselkedik, egynél nagyobb érték esetén viszont egyre inkább „széthúzódik”. Ezt a beállítást arra szokták használni, hogy a konvolúciós szűrők által érzékelt képrészlet méretét növeljék.



9.1. ábra. *A konvolúció végrehajtása egy kép tömbön több szűrővel (felül), és az ezzel ekvivalens konvolúciós réteg (alul).*



9.2. ábra. *A padding (felül), a stride (alul bal) és a dilatáció (alul jobb) hatása a konvolúció műveletére.*

9.2.2. Pooling

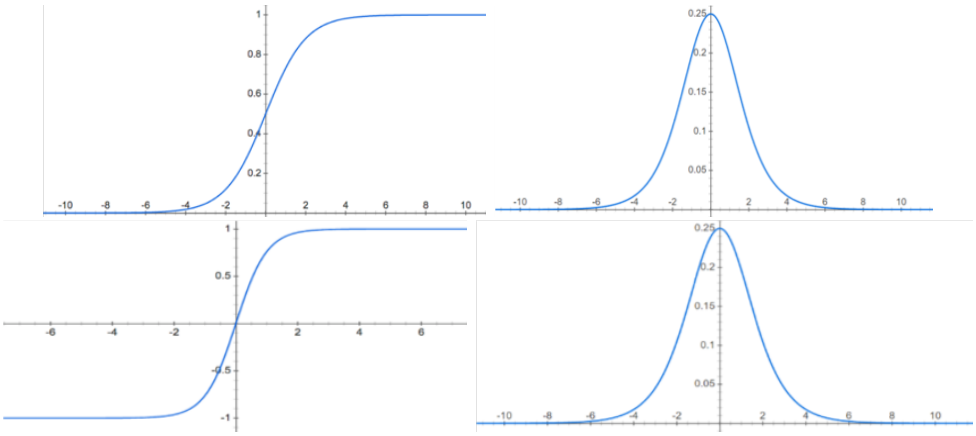
Érdemes észrevenni, hogy amennyiben egymás után újabb és újabb konvolúciós rétegeket helyezünk el, egyre növekvő mélységgel, akkor egy idő után a rétegek kimenetén kapott aktivációs tömb

mérete hatalmas lesz. Éppen ezért néhány rétegenként érdemes az aktivációs tömb térbeli méreteit csökkenteni. Az egynél nagyobb stride paraméterrel rendelkező konvolúciós réteg mellett ezt még az úgynevezett pooling operáció segítségével is megtehetjük. A pooling szintén egy csúszóablakos művelet, amely az aktivációs tömb éppen lefedett részét egyetlen számmal helyettesíti. A két leggyakoribb eset, amikor ez az érték az ablak által lefedett értékek átlaga vagy maximuma.

9.3. ábra. *A max (felül) és az átlag (alul) pooling.*

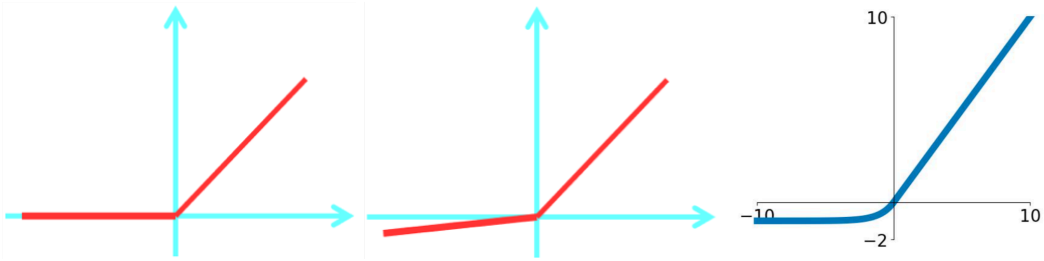
9.2.3. Aktivációk

A többrétegű neurális hálók utolsó esszenciális rétege az aktivációs réteg, ami tipikusan minden konvolúciós és lineáris réteget követ. Ez a két réteg ugyanis lineáris műveleteket hajt végre, ezek kompozíciója is lineáris marad. Éppen ezért az egyes rétegek közé nemlineáris függvényeket ékelünk be, amelyek az aktivációs tömb minden elemén függetlenül lefutnak. A hagyományos neurális hálók esetében népszerű választás volt a szigmoid, illetve a hiperbolikus tangens függvény. Ezen függvényeknek azonban közös hátránya, hogy az értelmezési tartományuk nagy részén a formájuk lapos, vagyis a deriváltjuk gyakorlatilag nulla. Ha sok ilyen deriváltat ékelünk a neurális hálóba, akkor a láncszabály értelmében előbb-utóbb a legtöbb deriváltat ki fogják nullázni. Ez a háló bemenethez közeli rétegeinek a beragadásához és a tanulás meghiúsulásához vezet.



9.4. ábra. *A sigmoid függvény és deriváltja (felül), valamint a tanh függvény és deriváltja (alul).*

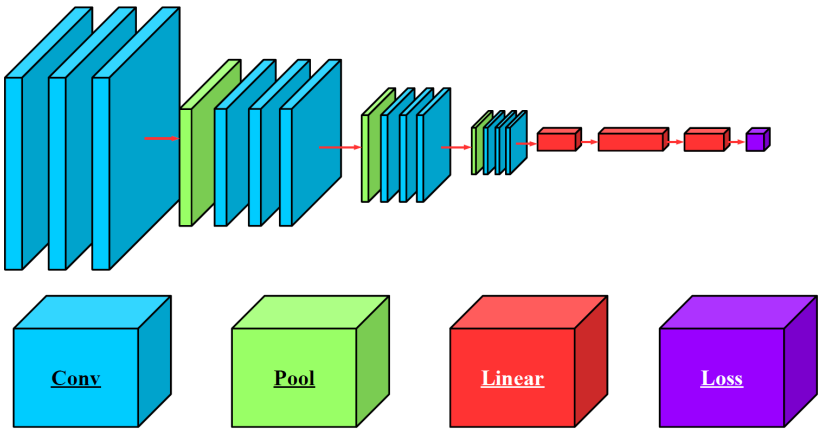
A jelenlegi hálók esetén a legnépszerűbb aktivációs függvény a ReLU (Rectified Linear Unit), ami egy szakaszosan lineáris aktiváció, amely a negatív bemeneteket kinullázza, míg a pozitív tartományban nem fejt ki hatást. Ennek az aktivációnak a deriváltja a pozitív tartományban 1, a negatívban 0, így a deriváltakra kifejtett zavaró hatása lényegesen kisebb. Ennek ellenére ReLU használata esetén is előfordulhat az elülső rétegek beragadása, amelyre a paraméteres ReLU (PReLU), valamint a szivárgó (Leaky) ReLU adhat megoldást. Ezek annyiban különböznek az eredeti megoldástól, hogy a negatív tartományban nem nullázzák az aktivációkat, hanem egy egynél kisebb konstanssal szorozzák azokat. A szivárgó esetben ez a konstans egy hiperparaméter, míg a paraméteres esetben a gradiens módszer segítségével tanulható. Létezik továbbá a ReLU egy olyan változata, amely a hagyományos verzióval ellentétben teljesen sima, így minden pontban deriválható. Ezt Elastic Linear Unit-nak (ELU) nevezzük.



9.5. ábra. *A ReLU (bal), Leaky ReLU (közép) és az ELU (jobb) aktivációk.*

9.3. Architektúrák

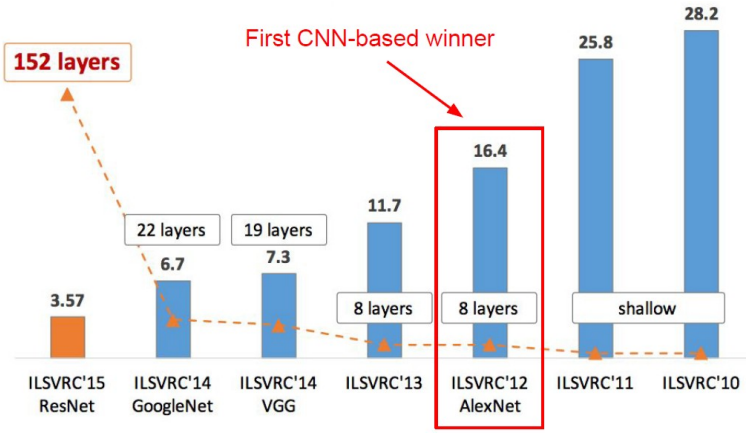
A jelen fejezetben bemutatott rétegeket tartalmazó neurális hálózatokat konvolúciós neurális hálózatoknak nevezzünk, melyeket első sorban a számítógépes látás területén alkalmaznak. A konvolúciós rétegek alkalmazása kifejezetten előnyös képi adatok esetén, mivel egy konvolúciós rétegnek a lineárishoz képest több nagyságrenddel kevesebb paramétere van. Egy konvolúciós neurális hálóban általában konvolúciós rétegek sorozata követi egymást (mindegyik kimenetén aktivációs függvénnyel), néhány konvolúciós rétegenként egy leskálázó réteg (konvolúció stride-dal, vagy pooling) közbe ékelésével. A háló végén tipikusan egy vagy több lineáris réteg állítja elő a végső kimenetet.



9.6. ábra. *Egy tipikus konvolúciós neurális háló.*

Érdemes elgondolkozni azon, hogy mit is csinál több egymással sorba kötött konvolúciós réteg. Egy konvolúció elképzelhető egy egyszerű jellemző detektorként, amely a bemeneteinek bizonyos kombinációira aktiválódik, míg másokra nem. Így az első konvolúciós réteg kimenetén kapott aktivációs térkép azt adja meg, hogy hol voltak olyan pixel kombinációk a képen, amik az egyes szűrőket aktiválták. A következő réteg bemenete azonban már ez az aktivációs térkép. Az ebben lévő szűrők tehát már nem a pixelek, hanem ezeknek az alacsonyabb szintű jellemzőknek bizonyos kombinációira aktiválódnak. Belátható ez alapján, hogy egy sokrétegű konvolúciós neurális háló kezdetben primitív képi jellemzők egyre bonyolultabb kompozícióit detektálni képes rétegeket tartalmaz a háló végső részeiben. Ez a szemlélet meglehetősen hasonlít az emberi látás kompozíciós jellegére.

Bár a legelső sikeres konvolúciós hálózatok egy ehhez hasonló architektúrát valósítottak meg, ezeknek számos, fejlettebb változata is létezik. Ezen fejlesztéseknek általában két alapvető motivációja van: az egyik, hogy a mély neurális hálók numerikusan problémás konvergencia tulajdonságait valamilyen módon javítsuk. A másik, hogy olyan struktúrákat alkossunk, amelyek minél kevesebb szabad paraméter mellett minél komplexebb összefüggéseket meg tudnak tanulni, ennek segítségével ugyanis a túlillesztés jelensége csökkenthető. A jelen fejezetben ezen architektúrák mögött rejlő motivációt ismertetjük.



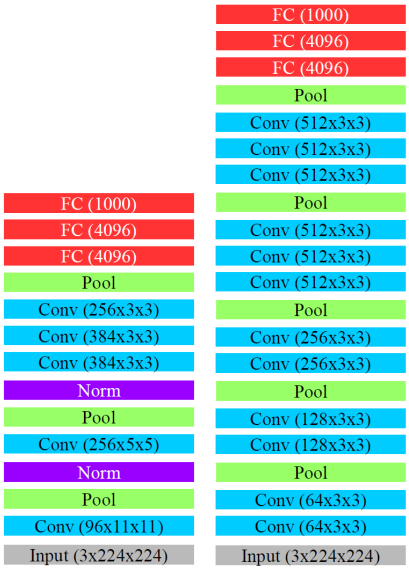
9.7. ábra. *Az ImageNet klasszifikációs verseny nyertes hálói és azok rétegszáma.*

9.3.1. AlexNet

Az egyik legelső sikeres konvolúciós háló architektúra az AlexNet volt, amely elsőként alkalmazott 10 körüli rétegszámot. Ennek a sikernek a legfőbb oka a ReLU aktivációs függvény és a normalizációs rétegek használata volt.

9.3.2. VGG

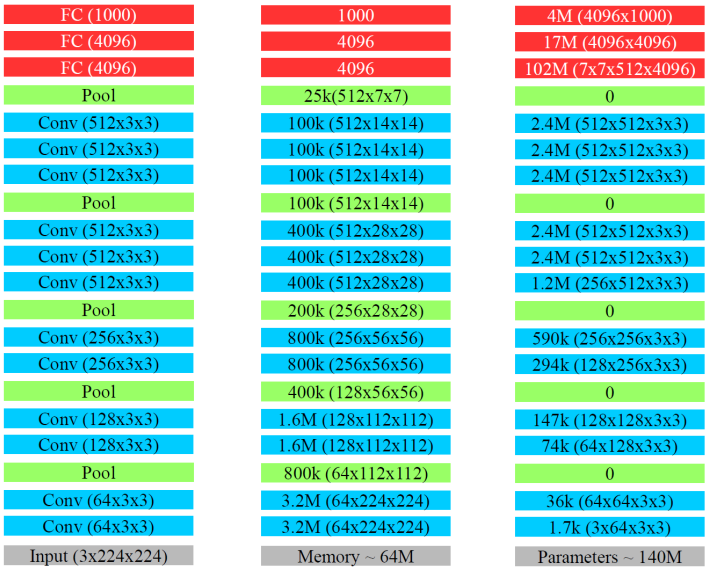
A VGG architektúra az egyik legnépszerűbb hagyományos neurális háló modell, melynek 16 és 19 rétegből álló változata is létezik. Az AlexNet-hez képest lényegesen szabályosabb architektúrája van, melyben kizárólag 3x3-as kernelméretű konvolúciós rétegeket alkalmaznak. E mögött a motiváció az, hogy három egymás mögé rakott 3x3 réteg ugyanakkora effektív látómezővel rendelkezik, mint egy darab 7x7, azonban kettővel több nemlinearitással és feleannyi paraméterrel rendelkezik. Ennek következtében összetettebb függvények megtanulása válik lehetővé, a kisebb paraméterszám viszont az overfitting valószínűségét csökkenti. Érdemes azonban megjegyezni, hogy ez nem jelenti, hogy a VGG architektúra különösebben jó lenne az overfitting elkerülésére, ugyanis a háló végén levő 3 lineáris réteg hatalmas paraméterszáma ezt az előnyt semlegesíti.



9.8. ábra. *Az AlexNet (bal) és a VGG (jobb) modellek.*

Az alábbi ábrán látható a VGG modell egyes rétegei számára szükséges memória mérete, és a réteg

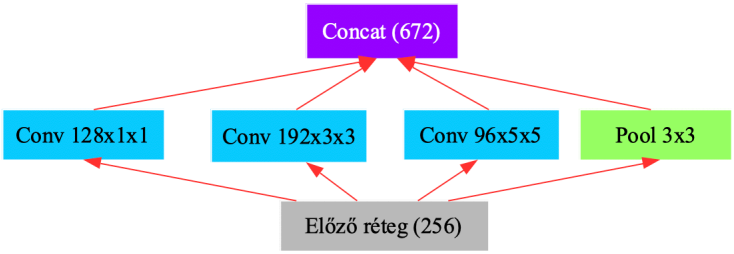
paramétereinek száma. Könnyen észrevehetjük, hogy a háló eleje a háló leginkább memóriaigényes része, paraméterek tekintetében viszont az utolsó rétegek szerepelnek igazán hangsúlyosan. Érdemes megjegyezni, hogy a VGG a modern hálótípusokhoz képest rendkívül pazarló mind paraméterszám, mind memóriahasználat tekintetében.



9.9. ábra. *A VGG háló által használt paraméterek száma és memória mérete.*

9.3.3. Inception

Paraméterek csökkentésére irányuló fejlesztésre jó példa az Inception névre hallgató réteg típus. Ennek a megoldásnak a lényege, hogy a konvolúciós rétegek nem csak sorban, hanem egymással párhuzamosan is létezhetnek. Ezek a párhuzamos rétegek általában különböző méretű konvolúciókat hajtanak végre, így egy olyan háló struktúrát eredményezve, amely lényegesen jobban tudja kezelni az objektumok skálájában fellépő variációkat.

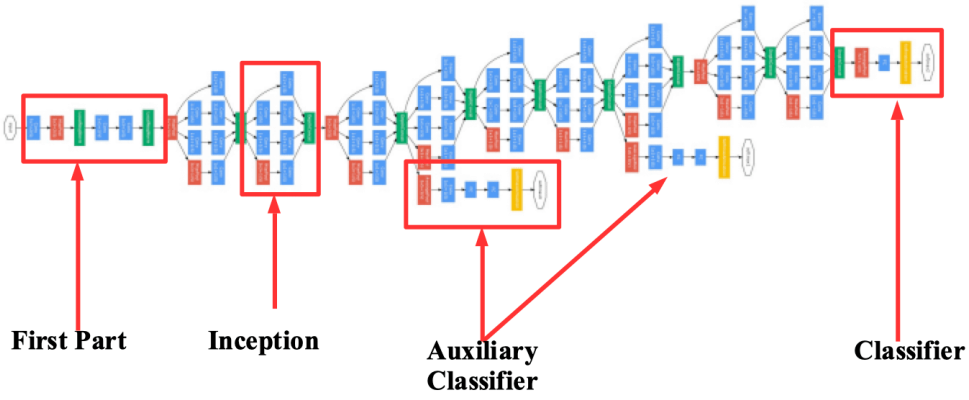


9.10. ábra. *A naiv Inception réteg.*

A GoogLeNet nevű hálózat számos Inception blokkból épül fel. Ezen felül a hálónak több alsóbb szintről nyíló segéd osztályozója van, melyeknek célja, hogy segítsék a konvergenciát a gradiensek alsóbb szintre történő bevezetésével.

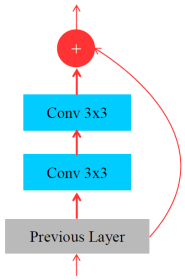
9.3.4. ResNet

A konvergencia javítására az úgynevezett reziduális blokk jó példa. Ez a réteg a konvolúció végrehajtása utáni kimenethez hozzáadja a bemenetet, így a rétegnek tulajdonképpen az elvárt be- és kimenet közti különbséget kell előállítania. Ennek a megoldásnak az a haszna, hogy az ilyen blokkokból álló neurális hálóban ezeken az összeadásokon keresztül a hiba deriváltja számára egy



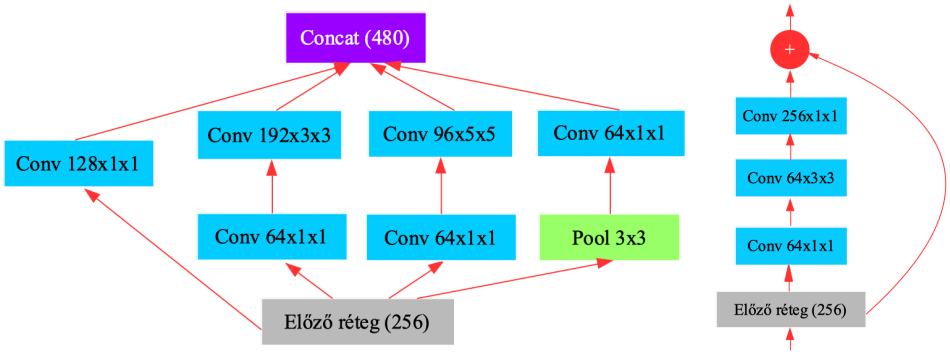
9.11. ábra. *A GoogLeNet modell.*

olyan út vezet vissza a háló elülső rétegeihez, ami mentén a derivált egy. Ennek következtében a hátraterjesztés során végrehajtandó számtalan szorzásból eredő numerikus problémák orvosolhatók. A reziduális blokk bevezetésének hatására a konvolúciós hálók maximális mélysége a 30 réteg körüli értékről a 100-200 réteg nagyságrendjére növekedett.



9.12. ábra. *A reziduális blokk.*

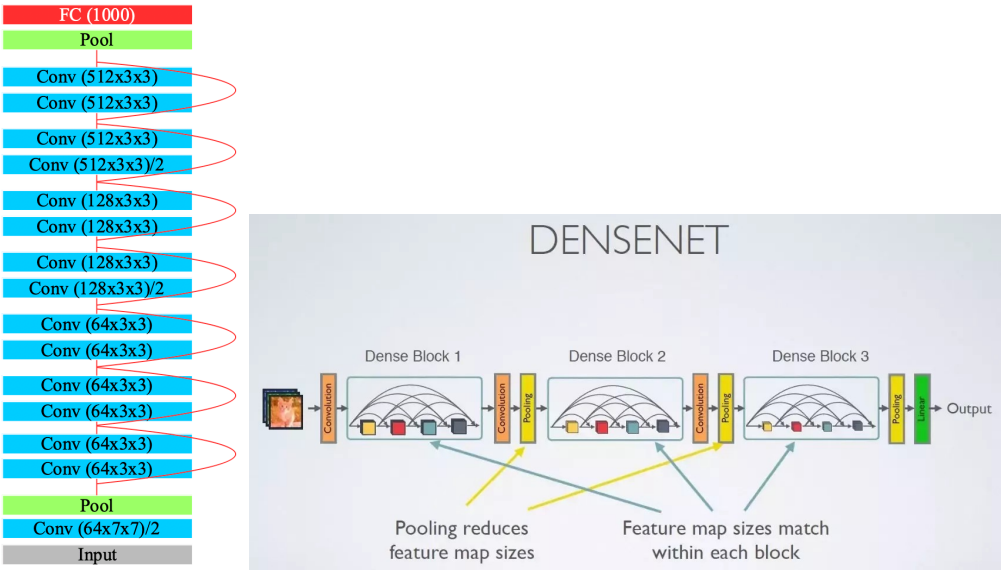
Érdemes még megemlíteni az úgynevezett tömörítő (bottleneck) rétegeket, amiket mind az Inception, mint a reziduális blokkok alkalmaznak. Ezek motivációja az, hogy a kétdimenziós konvolúciók elvégzése rendkívül drága, amennyiben az aktivációs térképek csatornáinak száma nagy. Éppen ezért ezekben a hálókban minden kétdimenziós konvolúciót megelőz egy 1x1-es konvolúciós réteg, amelyik a bemeneti aktivációs térkép csatornáinak számát annak töredékére csökkenti, vagyis tömöríti. Ezt követően a kétdimenziós (3x3, vagy 5x5) konvolúciót ezen a tömörített térképen végezzük el, ezt követően egy újabb 1x1-es konvolúciós réteg az eredeti csatornaszámot visszaállítja. Ezzel a módszerrel mind az egyes rétegek paramétereinek száma, mind a réteg végrehajtásának ideje nagymértékben csökkenthető.



9.13. ábra. *A Bottleneck megoldást alkalmazó Inception (bal) és reziduális (jobb) blokkok.*

9.3.5. DenseNet

A DenseNet architektúra a ResNet továbbgondolt változata, melyben egy dense blokkon belül nem csak a közvetlenül egymás utáni rétegek között találhatók áthidaló kapcsolatok, hanem egy blokkon belül minden rétegpár között (innen a dense elnevezés). Ez az újítás lehetővé tette, hogy a ResNet modellel ekvivalens eredményeket érjenek el hozzávetőlegesen feleannyi számítás és paraméter árán.



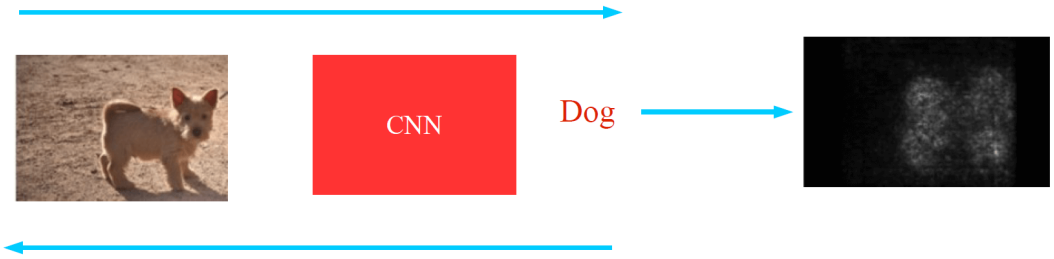
9.14. ábra. *A ResNet (bal) és a DenseNet (jobb) modellek.*

9.4. Vizualizáció

A korábbi és a következő részekben részletesen tárgyaljuk a különböző mély neurális hálók felépítésének, tanításának, validációjának és installálásának kérdéseit. Egy rendkívül fontos dologról azonban nem esett szó: ez pedig a hibakeresés problémája. Ez a neurális hálók esetében kifejezetten kritikus kérdés, hiszen a próbálkozás módszere rendkívül költséges, ugyanis egy nagyobb háló tanítása napokig, vagy akár hetekig is eltarthat. A probléma azonban az, hogy a neurális hálók egyfajta fekete dobozként viselkednek, vagyis nem nagyon értjük, hogy pontosan mit és miért csinálnak odabenn. Éppen ezért szükséges lehet a neurális hálók belső működésének valamilyen jellegű vizualizációjára, aminek segítségével betekintést nyerhetünk a belső állapotokba.

Érdemes észrevenni, hogy a gradiens számolás során alkalmazott hátraterjesztés művelete valójában általánosságban felhasználható a háló két tetszőleges pontja közti derivált számítására. Ez az észrevétel számos különböző módon felhasználható. Egyrészt lehetővé teszi annak meghatározását, hogy melyik pixelek befolyásolják egy osztály jóság értékét egy adott képen. Ez felhasználható az objektumok szegmentálására, követésére, valamint arra is, hogy egy helytelen osztályozás esetén megtudjuk, hogy a képen melyik rész felelős a hibáért.

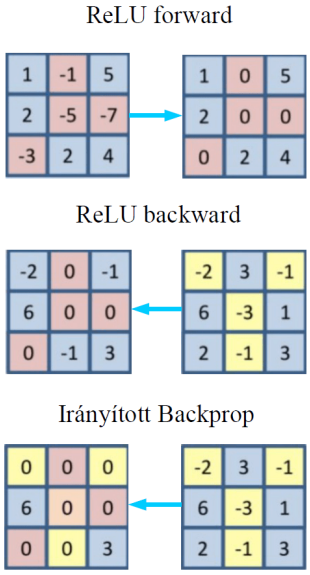
Ehhez semmi mást nem kell tennünk, csak a kimeneten a hibafüggvény gradiense helyett a kiválasztott osztály jóság kimeneti gradiensére egy értéket írunk elő, míg az összes többire nullát. Ezt követően végrehajtjuk a hátraterjesztés műveletét, azonban ebben az esetben nem a súlyok szerinti, hanem a bemeneti kép szerinti deriváltat határozzuk meg. Ennek az abszolút értékét véve, majd a csatorna dimenzió mentén maximumot véve előállíthatunk egy szürkeárnyalatos képet, ahol a pixelek intenzitása az adott jóság értékre kifejtett hatással arányos. Az így kapott képet saliency-nek nevezzük.



9.15. ábra. *A saliency előállítása.*

9.4.1. Guided backpropagation

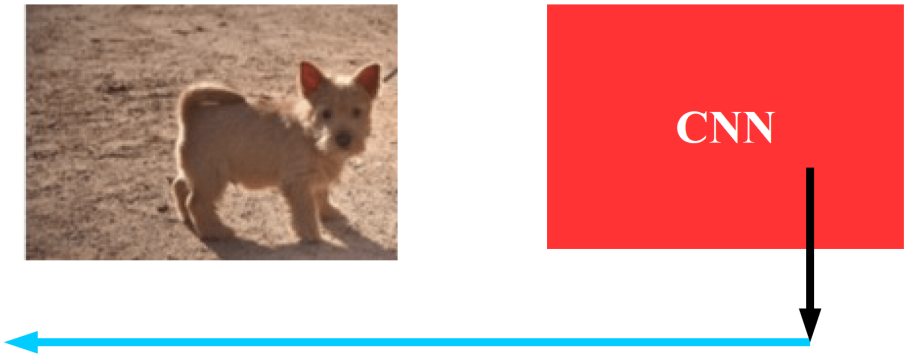
Ha szeretnénk ezt a módszert szegmentálásra vagy vizualizációra használni, akkor azonban szembesülünk egy apróbb problémával: a saliency ugyanis azokon a képrészleteken is nagy lesz, amik a kimenetet negatívan befolyásolták (vagyis "ahol nincs kutya" például). Ennek kiküszöbölésére a hátraterjesztés során el kellene nyomni azoknak a jellemzőknek a hatását, amelyek a vizsgálni kívánt osztály ellen hatnak. Ezt könnyedén megtehetjük, ugyanis könnyedén belátható, hogy ha egy adott jellemző a vizsgált végeredmény hatását csökkenti, akkor a hozzájuk tartozó derivált érték negatív lesz. Innentől kezdve csak annyi dolgunk van, hogy a hátraterjesztés során ezeket a gradienseket minden réteg esetén nullázzuk. Ezt a legegyszerűbb úgy megoldani, hogy a ReLU backward lépését úgy módosítjuk, hogy magukra a gradiensekre is alkalmazzuk a ReLU függvényt. Ezt a műveletet hívjuk irányított, azaz guided backpropagationnek.



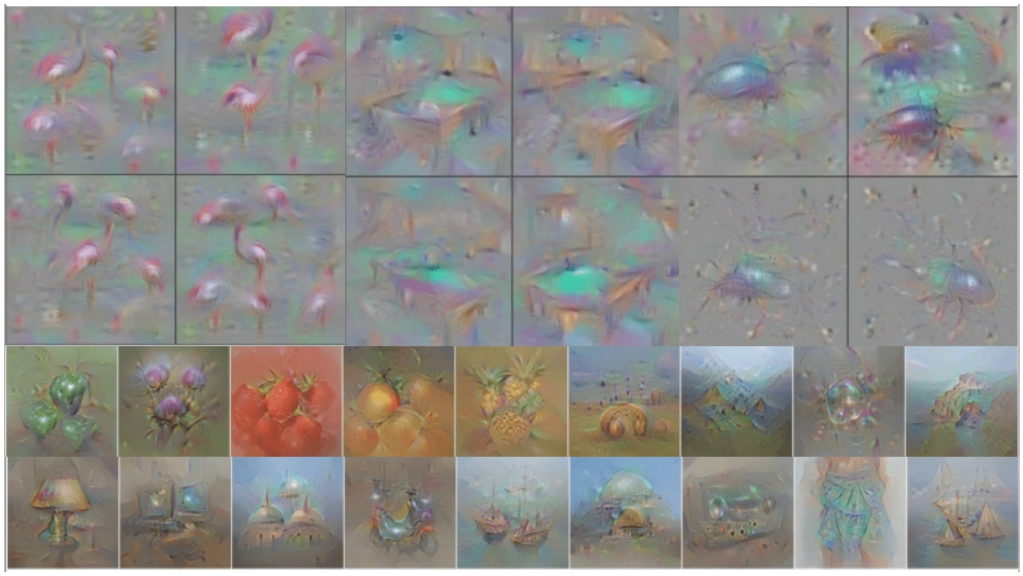
9.16. ábra. *A guided backpropagation alkalmazása a ReLU nemlinearitáson.*

A jó minőségű saliency előállításán felül a guided backpropagation felhasználható tetszőleges neuronok vizualizációjára. Ekkor a célunk, hogy előállítsuk azt a képet, ami az adott neuront (értsd: konvolúciós szűrőt) maximálisan aktiválja. Ehhez először csupa nullával inicializáljuk a bemeneti képet, majd azt a hálóban a kiválasztott neuront tartalmazó rétegig előreküldjük. Ezt követően a réteg kimeneti gradienseit a korábban ismertetett módon előírjuk, és elvégezzük a backpropagation műveletét egészen a képig. Ezután a kapott gradienseket hozzáadjuk a kép pixeleihez, az előző két lépést addig ismételve, amíg a kép számottevően változik.

Fontos megjegyezni, hogy a kapott képen számos javítást kell végeznünk, különben értelmezhetetlen lesz, valamint a pixel értékek folyamatosan nőni fognak és az algoritmus sosem fog leállni. Emiatt szükségszerű a képen valamilyen (általában L2) regularizációt végrehajtani, valamint rendszeresen simító szűrők segítségével zajt szűrni. Érdemes továbbá a backpropagation során kapott túlságosan kicsi pixel és gradiens értékeket kézzel nullázni.



9.17. ábra. *A guided backpropagation elve.*



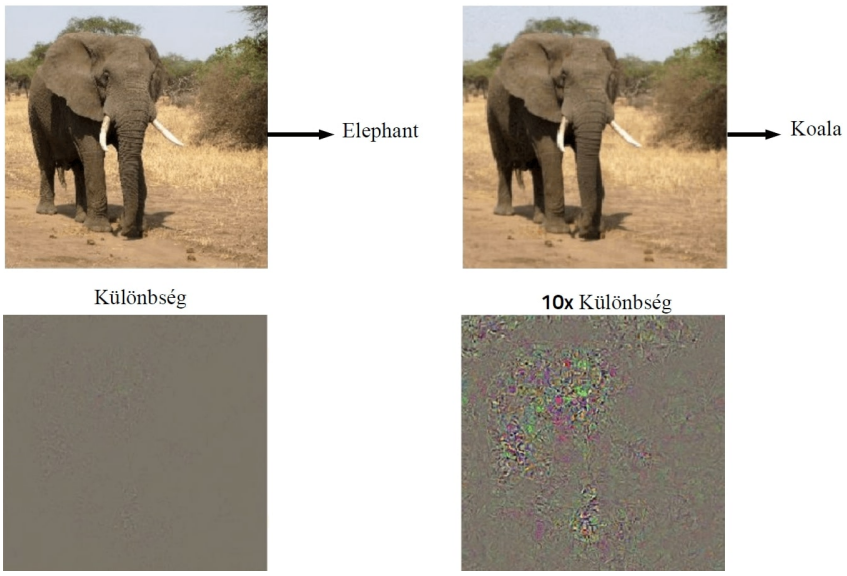
9.18. ábra. *A guided backpropagation eredménye.*

9.4.2. Ellenséges példák

A backpropagation algoritmus alkalmazásai során egy fontos felfedezés volt, hogy egy tipikus számítógépes látásban használt neurális háló esetén lehetséges helyesen osztályozott képeken olyan minimális, emberi szem által észrevehetetlen változásokat generálni, aminek hatására a neurális háló már tévesen fogja az adott képet osztályozni. Ezeket a képeket ellenséges példának nevezzük, és jelenlegi tudásunk szerint meglehetősen nehéz ellenük védekezni. A legjobb, amit tehetünk az, hogy a tanítás során magunk generálunk ilyen példákat, és ezekkel is tanítjuk a hálót. Persze az ember sem mentes ilyen hibáktól – az emberi látás ellenséges példáit illúzióknak nevezzük. Úgy tűnik azonban, hogy a neurális hálók illúziói jelentős mértékben különböznek az emberétől.

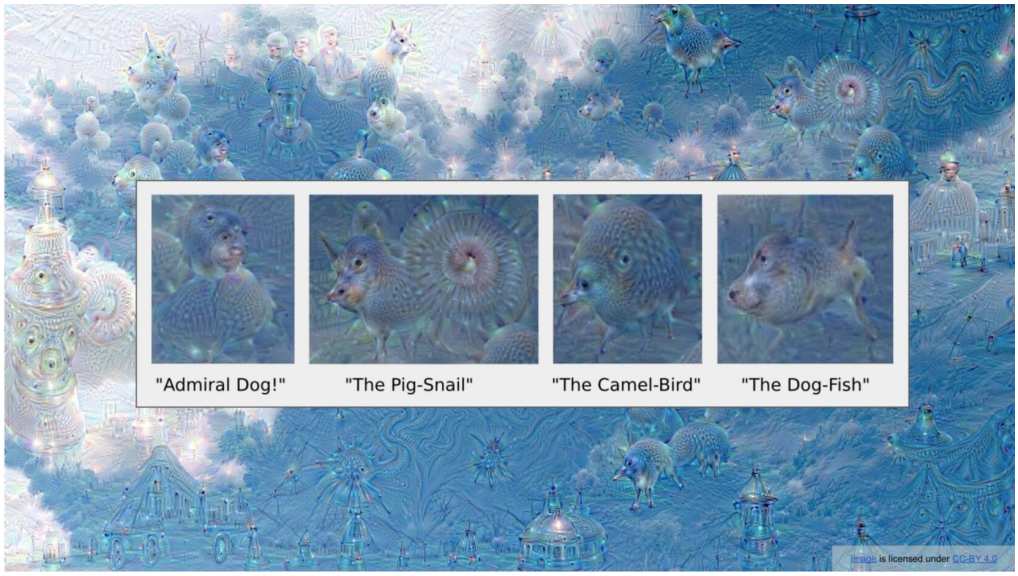
9.4.3. DeepDream

A guided backpropagationnek létezik még egy érdekes alkalmazása. Ez az alkalmazás annyiban különbözik a vizualizációtól, hogy a bemeneti kép nem üres, hanem egy tetszőleges valós kép. Ezt követően ezt ugyanúgy előreküldjük egy kiválasztott rétegig, azonban a réteg kimeneti gradiensét másképp írjuk elő. A korábban alkalmazott one-hot vektor helyett a kimeneti gradienst egyszerűen egyenlővé tesszük magával a réteg aktivációjával. Ez egyszerűen megfogalmazva annyit fog eredményezni, hogy azokat a jellemzőket, amiket a háló a képen erősen detektált felerősítjük, amiket pedig nem látott, tovább gyengítjük. Más szóval létrehozunk egyfajta pozitív visszacsatolást a kép és az adott réteg aktivációi közt. Az eredmény egy meglehetősen kreatív, (rém)álomszerű kép.



9.19. ábra. *Az ellenséges (adversarial) példák.*

Bár ennek a módszernek a gyakorlati haszna elenyésző, bizonyította azonban, hogy konvolúciós neurális hálók rendelkezhetnek az emberihez hasonló asszociációs készségekkel.



9.20. ábra. *A deep dream eredménye.*

További Olvasnivaló

[18] J. Heaton, "Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning", *Genetic Programming and Evolvable Machines*, 19. évf., 1-2. sz., 305–307. old., 2017. okt. DOI: 10.1007/s10710-017-9314-z. cím: https://doi.org/10.1007%2Fs10710-017-9314-z.

[21] C. Szegedy, W. Liu, Y. Jia és tsai., "Going deeper with convolutions", *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2015. jún. DOI: 10.1109/cvpr.2015.7298594. cím: https://doi.org/10.1109%2Fcvpr.2015.7298594.

[22] K. He, X. Zhang, S. Ren és J. Sun, "Deep Residual Learning for Image Recognition", *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2016. jún. DOI: 10.1109/cvpr.2016.90. cím: https://doi.org/10.1109%2Fcvpr.2016.90.

[23] G. Huang, Z. Liu, L. V. D. Maaten és K. Q. Weinberger, "Densely Connected Convolutional Networks", *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2017. júl. DOI: 10.1109/cvpr.2017.243. cím: https://doi.org/10.1109%2Fcvpr.2017.243.

[24] L. A. Gatys, A. S. Ecker és M. Bethge, *A Neural Algorithm of Artistic Style*, 2015. eprint: 1508.06576. cím: http://www.arxiv.org/abs/1508.06576.

[25] I. J. Goodfellow, J. Shlens és C. Szegedy, *Explaining and Harnessing Adversarial Examples*, 2015. eprint: 1412.6572. cím: http://www.arxiv.org/abs/1412.6572.

10. fejezet

Deep Learning a gyakorlatban

10.1. Bevezetés

A korábbi előadások során megismertük a mély tanulás és a konvolúciós neurális hálók alapjait, a következő fejezetben pedig részletesen tárgyaljuk majd a sorozatok feldolgozására, valamint az osztályozásnál bonyolultabb látási feladatok elvégzésére létrehozott speciális struktúrákat. A mély tanulás azonban tipikusan azon területek közé tartozik, ahol a módszerek használata papíron rendkívül egyszerűnek tűnik, a gyakorlatban viszont számtalan nehézség adódik, melyek a módszerek használatát nehézzé teszik. A jelen előadás célja, hogy összeszedje azokat a gyakorlati megfontolásokat és praktikákat, amelyek nélkül rendkívül nehéz a való életben jól működő neurális hálókat létrehozni.

A neurális hálók tanítását alapvetően négy dolog nehezíti meg:

1. Numerikus problémák, melyek az optimalizáló algoritmus konvergenciáját hiúsítják meg
2. A túlillesztés (overfitting) jelensége, amely a betanított modell való életben történő alkalmazhatóságát veszélyezteti
3. A hálók tanításához szükséges nagy mennyiségű címkézett adathalmaz előállításának problémája
4. A tanításhoz és a jó általánosításhoz vezető hiperparaméter értékek meghatározása

10.2. Konvergencia problémák

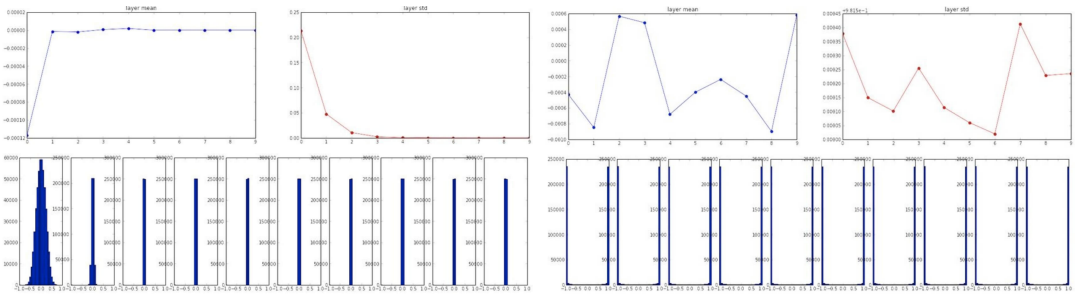
A korábbi fejezetben ismertetett gradiens módszer és backpropagation algoritmus a táblán és a tankönyvek (és ezen jegyzet) hasábjain minden esetben tökéletesen működik. A gyakorlatban azonban a lebegőpontos számábrázolás véges pontossága és tartománya miatt számos problémába ütközünk, amik az algoritmus konvergenciáját könnyedén meghiúsíthatják. Az esetek többségében elmondható, hogy a lebegőpontos aritmetika akkor működik igazán stabilan, ha a számaink eloszlása ”szép”, azaz nulla középértékű, és megközelítőleg egy szórású.

Ennek oka, hogy a backpropagation során valójában egy sor mátrixszorzást kell elvégeznünk. Könnyedén beláthatjuk, hogy amennyiben sok számot szorzunk össze, akkor amennyiben ezek a számok 1-nél kisebbek, egy idő után nullát kapunk, ha pedig egynél nagyobbak akkor végtelent. Ekkor a gradiens értékek (főleg a háló első rétegeinél, ahol a mátrixszorzat már meglehetősén hosszú) vagy kinullázódnak, vagy pedig elszállnak. Ezt hívjuk az eltűnő vagy a felrobbanó gradiensek problémájának. A szorzat csak akkor marad a véges tartományban, ha a számaink aránylag közel vannak egyhez.

Mátrixok szorzása esetében ez ugyanígy igaz, csak itt a mátrixok normájának kell egy körülinek lennie. A mátrixok esetében az egyik legelterjedtebb norma fajta az úgynevezett Frobenius norma, amely egyszerűen a mátrix elemeinek négyzetösszege. Amennyiben a mátrix elemeinek átlaga nulla, akkor ez megegyezik az elemek szórásnégyzetével.

10.2.1. Inicializáció

A mély tanulásról szóló első alfejezetben bevezettük a gradiens módszert, ami a hibafüggvény deriváltjának segítségével iteratívan módosította a háló paramétereit a teljesítmény javításának érdekében. Arról viszont szándékosan hallgattunk, hogy hogyan kapjuk meg a kezdeti paraméter értékeket. A helyzet az, hogy a problémát helyesen megoldó paraméterekről kezdetben nem tudunk semmit, így nincs más választásunk, mint véletlenszerűen inicializálni őket. Az viszont egyáltalán nem mindegy, hogy milyen szórású véletlen számokkal végezzük ezt el (a nulla középérték nyilvánvaló módon adja magát). Ha ugyanis a véletlen súlyok értéke túl nagy, akkor a legtöbb aktiváció értéke is egyre nagyobb lesz, melynek következtében a gradiensek is rendkívül nagyok lesznek. Ez szemléletesen azt fogja eredményezni, hogy az optimalizálás során nem kis lépésekben fogjuk a hiba minimumát közelíteni, hanem hatalmas ugrásokkal fogunk a paramétertérben haladni, jó eséllyel teljesen átugorva a minimum helyét. Túl kicsi súlyok esetében néhány réteg után a háló aktivációi szinte mindenhol közel nullák lesznek, melynek következtében a háló gradiensei is, így a háló a kezdeti értékekbe beragad.



10.1. ábra. *A háló rétegeinek aktivációinak eloszlása kicsi véletlen (bal) és nagy véletlen (jobb) számokkal történő inicializálás esetén.*

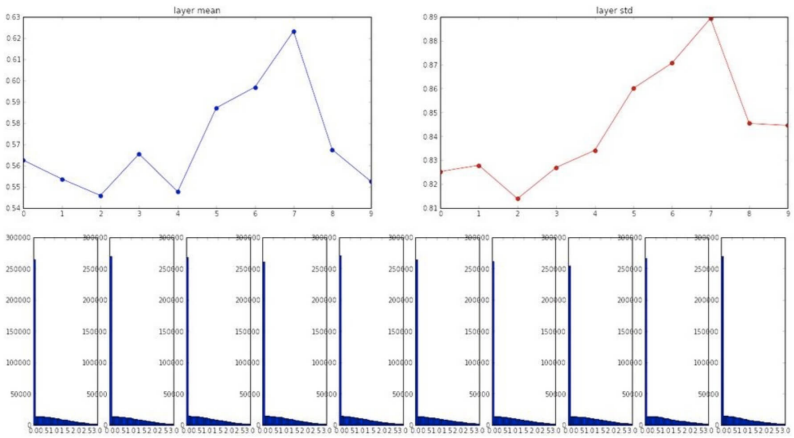
Ennek elkerülésére a háló súlyainak szórását nagy körültekintéssel kell megválasztani, hogy se túl kicsik, se túl nagyok ne legyenek. Ennek több módja is létezik, melyek közül a leginkább elterjedt választások a Xavier, illetve a He inicializációs formulák, amelyek a következőképp határozzák meg a háló egyes rétegeinek kezdeti súlyainak szórását:

$$\begin{aligned} \mu &= 0 \\ \sigma_{Xav} &= \frac{2}{n_i + n_o} \\ \sigma_{He} &= \frac{2}{n_i} \end{aligned} \tag{10.1}$$

Ahol n_i és n_o az adott réteg be- és kimeneteinek számát jelöli. Érdemes megjegyezni, hogy ezekkel a választásokkal a háló aktivációi és gradiensei megközelítőleg normális eloszlásúak lennének, azonban a ReLU aktivációs függvény használata ezt valamelyest torzítja.

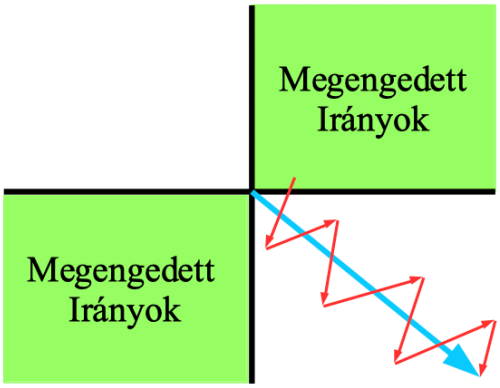
10.2.2. Adatnormalizálás

Hasonló megfontolások miatt szükséges a neurális hálóknak bemenetként szolgáló képeken bizonyos transzformációk elvégzése. A korábbiak alapján belátható, hogy a jó numerikus konvergencia érdekében rendkívül fontos, hogy a bemenetre adott kép pixel értékeinek eloszlása megközelítőleg sztenderd normális legyen. Hiába inicializáljuk ugyanis jól a háló súlyait, ha a bemenet értékei



10.2. ábra. *Az aktivációk eloszlása Xavier-inicializáció esetében.*

túlságosan nagy számok, akkor hasonló problémába fogunk ütközni, mint a túl nagy súlyok esetén. Szintén fontos, hogy a pixelek átlaga a nulla közelében legyen, ugyanis csupa pozitív, vagy csupa negatív bemenet esetén a gradiens lehetséges irányait korlátozzuk.



10.3. ábra. *Csupa pozitív bemenet esetén a gradiens is csupa pozitív, vagy csupa negatív lesz, így csak "cikkcakkban" képes a kívánt irány felé haladni.*

10.3. Validáció és regularizáció

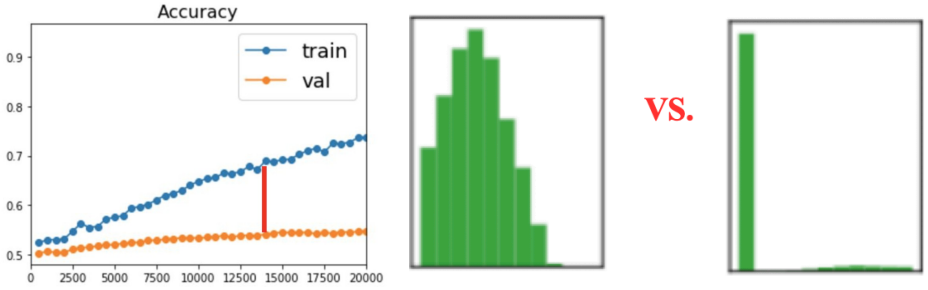
A gépi tanulás alapjainál említettük, hogy a tanítás során felléphet az overfitting jelensége. Ennek észleléséhez két külön adathalmazt érdemes használunk, egy tanító és egy validációs részt. Ezek lehetnek ugyanannak az adatbázisnak a véletlenszerűen kiválasztott részei (arányuk általában 80%-20% között mozog). A tanító szett segítségével végezzük el a tanítást, míg a validációs szett segítségével az overfitting jelenségét monitorozzuk, melynek alapján a hiperparaméterek hangolhatók. Fontos hangsúlyozni, hogy a validációs adatbázist **SOHA** nem használjuk tanításra.

A gyakorlatban azonban két adatbázis nem elég. A tanító adathalmazt ugyanis a háló paramétereinek meghatározásához, míg a validációs adatbázist a háló hiperparamétereinek hangolásához használjuk. Így viszont nem tudjuk azt megmondani, hogy teljesen új adatokon milyen pontossággal teljesít majd a hálózat. Ehhez egy harmadik, a teljes adatmennyiség hozzávetőlegesen 10%-át kitevő teszt adatbázist érdemes használni. A módszer megbízhatóságához elengedhetetlen, hogy ezt a három adatbázist teljesen elkülönítsük és csak egyetlen célra használjuk.

Érdemes megjegyezni, hogy a neurális hálók túlillesztésének jelensége szintén jellemezhető az egyes rétegek aktivációinak eloszlásával. A túlillesztés esetén ugyanis az történik, hogy a háló a be- és

10.4. ábra. *Az adatbázisok elosztása.*

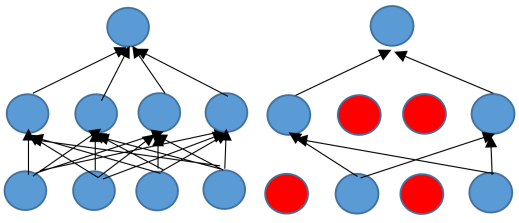
kimenetek közötti általános összefüggések helyett az egyes bemeneti tanítópéldákra adandó helyes választ kezdi el egyesével megtanulni. Ez tipikusan azt jelenti, hogy az egyes rétegekben minden egyes tanító adat esetén csak nagyon kevés aktiváció lesz maximális (amelyek épp az adott tanító példára emlékeznek), míg a többi aktiváció éppen az ellenkező véglet értékét veszi fel. Amint azt az imént beláttuk, ilyen „végletes" aktivációk akkor tudnak könnyedén előfordulni, ha az egyes rétegek súlyai túlságosan nagyra nőnek. Ha visszaemlékezünk a korábban tárgyalt regularizációs módszerekre, azok pont a súlyok növekedését próbálták fékezni.



10.5. ábra. *Az overfitting jelenségéhez tartozó tanítási és validációs görbék (bal), valamint az aktivációk eloszlása normális (közép) és overfitting (jobb) esetben.*

10.3.1. Dropout

A nem kívánt aktivációk elkerülésére két további gyakran alkalmazott módszer létezik. Ezek közül az első a dropout nevű eljárás, melynek lényege, hogy a tanítás során minden egyes előreterjesztés során az egyes rétegek aktivációinak bizonyos hányadát véletlenszerűen kinullázzuk, és a további rétegek aktivációit így számoljuk tovább (10.6. ábra). Könnyen belátható, hogy ez a módszer meglehetősen csökkenti a túlillesztés mértékét, hiszen a hálót ezzel a módszerrel redundanciára kényszeríti. Fontos megjegyezni, hogy a tesztelés során a véletlenszerű törléseket már nem végezzük el, így viszont az egyes aktivációkat a dropout valószínűségének arányában skálázni kell.



10.6. ábra. *A dropout alkalmazása.*

10.3.2. Batch Normalization

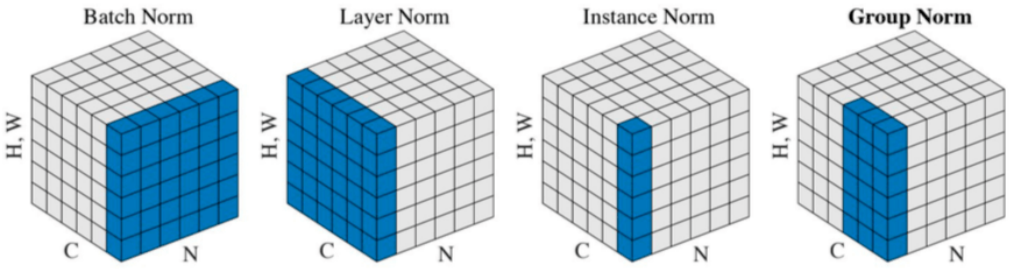
A másik megoldás az úgynevezett batch normalizálás, aminek lényege, hogy az egyes rétegek után egy normalizáló műveletet végzünk el. Korábban említettük, hogy a tanítás során egyszerre nem egy képet, hanem egy úgynevezett minibatch-nek megfelelő (általában 32 többszöröse) képet értékelünk ki. A batch normalizálás műveletének lényege, hogy az egyes aktivációk átlagát és szórását a tanítás során folyamatosan számoljuk, és az egyes aktivációkat ennek segítségével normalizáljuk.

Érdemes megjegyezni, hogy ez a művelet nem csak a túlillesztést mérsékeli az aktivációk eloszlásának normalizálásával, hanem a numerikus konvergenciát is javítja. A batch normalizálás képlete az alábbi:

$$\begin{aligned} x_{BN} &= \frac{x-\mu}{\sigma^2+\epsilon} \quad \leftarrow \quad vanilla \\ x_{AF} &= \alpha x_{BN} + \beta \quad \leftarrow \quad affin \\ x_{DC} &= \Sigma^{-\frac{1}{2}}(x-\mu) \quad \leftarrow \quad decorrelated \end{aligned} \tag{10.2}$$

Ahol μ és σ/Σ az x bemenetek becsült átlaga és szórása/kovariancia mátrixa, míg α és β tanult paraméterek. Érdemes megjegyezni, hogy modern neurális hálókban a batch normalizálás teljesen alapvető művelet, így általában minden konvolúciós réteget követ egy ilyen réteg. A batch normalizálás és a dropout akár együttesen is használható, bár a legtöbb kísérlet minimális teljesítménynövekedést mutat csak. A batch normalizálás előnyei az alábbi három pontban foglalhatók össze:

- Az aktivációk eloszlásának normalizálása csökkenti az overfitting mértékét.
- A normalizálás miatt az egyes rétegek eloszlásai és gradiensei megközelítőleg ugyanabban a nagyságrendben mozognak, ami segíti a konvergenciát.
- Mivel a gradiens módszer során minden réteg súlymátrixát egyszerre változtatjuk, az első kivételével az összes réteg bemeneteinek eloszlása megváltozik minden lépésben, aminek utána kell tanulni. A batch normalizálás pont ezt küszöböli ki, így lényegesen stabilabbá (és következésképp gyorsabbá) teszi a konvergenciát.



10.7. ábra. *Érdemes megjegyezni, hogy a batch normalizáción kívül létezik még réteg (layer) normalizáció, ahol a teljes aktivációs tömböt normalizáljuk egyedenként külön. Egyed (instance) normalizáció esetén az egyes csatornákat külön-külön normalizáljuk a térbeli dimenziók mentén. A kettő közti kompromisszum a csoport normalizálás, ahol néhány csatornát egybe veszünk.*

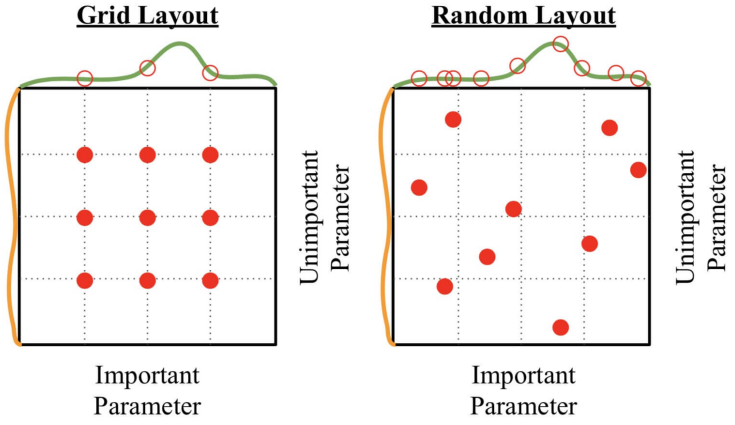
10.3.3. Adat Augmentáció

A túlillesztés jelenségének van még egy lehetséges elkerülési módja: gondoljunk bele, hogy a túlillesztés esetén a neurális háló a tanító adatbázis elemeire egyesével tanulja meg a helyes választ. Nyilvánvaló, hogy minél több tanító adat áll rendelkezésre, annál nehezebb ezt megtenni. Éppen ezért a tanító adatok számának növelése szinte minden esetben segít a túlillesztés mértékének csökkentésében. Tanító adatokat előállítani azonban rendkívül költséges, így ez a stratégia önmagában nem feltétlenül célravezető. Képek esetében azonban van lehetőségünk (részben) új tanítóadatok automatikus generálására, vagyis adat augmentációra. A módszer lényege, hogy tükrözés, véletlenszerű kivágás, forgatás, skálázás, intenzitástranszformációk segítségével mesterségesen növeljük az adatbázis méretét. Könnyen belátható, hogy ezek a műveletek a képek címkéjét nem befolyásolják, így büntetlenül elvégezhetők. Fontos megjegyezni, hogy a batch normalizálás, regularizáció és adat augmentáció módszereit együtt használjuk.

10.4. Hiperparaméter optimalizáció

A neurális hálók tanításának további nehézsége, hogy egy tipikus tanítás során több tucat hiperparaméter is rendelkezésünkre állhat, melyek megfelelő értékének megválasztására nincs a próbálkozásnál jobb módszerünk. Egy neurális háló tanítása azonban meglehetősen sokáig tarthat (néhány órától akár néhány hétig is), így minden egyes próbálkozás rendkívül költséges. Ezért a tanítás kezdetén számos hiperparaméter megközelítőleg helyes értéke megválasztható úgy, ha a tanítást a teljes adatbázisnak csak egy kis részén végezzük el. Ez a módszer az esetleges programhibák felderítésében is segítségünkre lehet.

A teljes adatbázison történő tanítás során általában már csak néhány hiperparaméter értékét kell egy aránylag szűk tartományon belül meghatározni. Ekkor célszerű lehet ezeket a tartományokat egy egyenletes rácsra osztva az egyes rácspontokban különböző tanításokat végezni, majd ezeket összehasonlítani. Ennél azonban sokkal célszerűbb, ha az előbbi módszerrel megegyező mennyiségű véletlen hiperparaméter kombinációkkal végezzük a tanítást. Ekkor ugyanis minden hiperparaméter esetében nagyobb felbontáson mérjük az adott paraméter hatását. Ez különösen abban a gyakori esetben hasznos, amennyiben a hiperparaméterek közül az egyik sokkal nagyobb mértékben befolyásolja a tanítás minőségét, mint a többi.



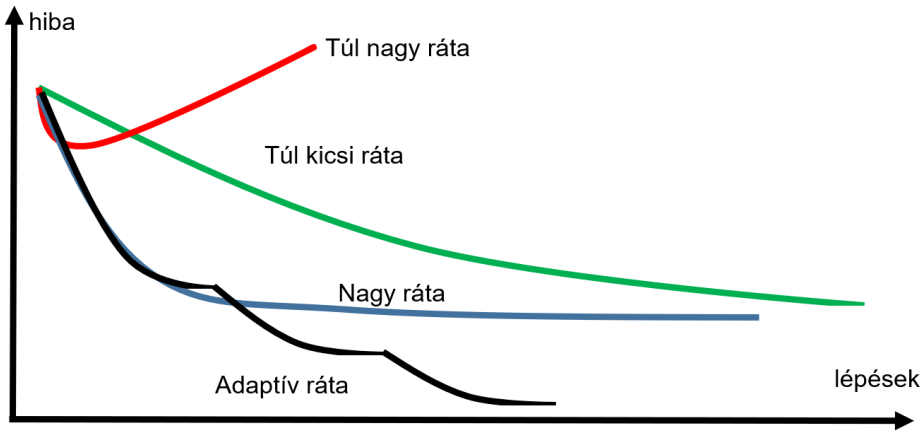
10.8. ábra. *A hiperparaméterek optimalizációjának két lehetséges sémája.*

10.4.1. Tanulási ráta

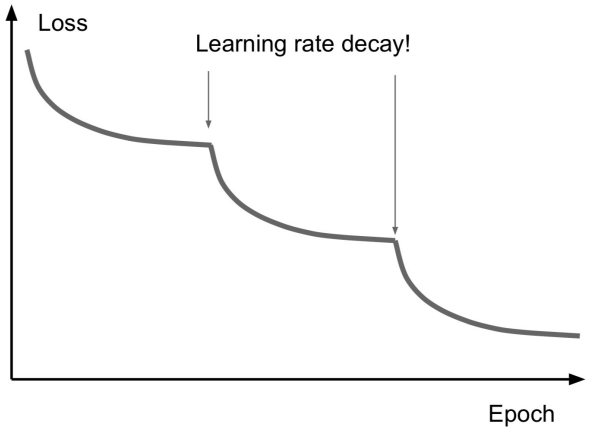
A tanítás hiperparaméterei között külön tárgyalást igényel a gradiens módszer lépéseinek nagyságát meghatározó tanulási ráta. A gradiens módszer során a hibafüggvény által képzett „völgy” legmélyebb pontjába szeretnénk a legmeredekebb csökkenés irányába tett lépések sorozatával eljutni. Amennyiben a lépések mérete túlságosan kicsi, akkor csak nagyon sok lépés után jutunk be a völgybe. Ha azonban a lépések mérete túl nagy, akkor ugyan gyorsan eljutunk a legmélyebb pont közelébe, az utolsó lépéssel azonban átlépünk a völgyön, és onnantól kezdve az idők végezetéig a völgy két oldala között fogunk „pattogni”. Sőt óriási lépésméret esetén előfordulhat, hogy akkorát ugrunk a völgy közepe felé, hogy annak ellenkező oldalán magasabb helyre lépünk, mint korábban voltunk. Ha ezt ismételgetjük, akkor minimalizálás helyett kimászunk a völgyből.

A gyakorlatban ezen megfontolások miatt nem egyetlen tanulási rátát szokás alkalmazni. E helyett a tanítás kezdetén a legnagyobb olyan tanulási rátával indítjuk az optimalizálást, amivel a hiba értéke stabil csökkenést mutat, így a lehető leggyorsabban jutunk az optimum közelébe. Egy idő után a ráta értékét csökkentjük, így engedve, hogy a valódi optimum pozícióját minél jobban megközelítsük. Ez felfogható egyfajta durva optimalizálási és finomhangolási lépésként. A tanulási ráta állítására sok lehetséges módszer létezik, melyek közül az egyik leggyakoribb a ráta fix faktorral történő csökkentése bizonyos számú lépés után.

Lehetőség van természetesen a ráta adaptív állítására a tanulási sebesség folyamatos monitorozása által. Ekkor a rátát akkor csökkentjük egy fix faktorral, amennyiben a tanulás már bizonyos



10.9. ábra. *A különböző tanulási ráta választások hatása.*



10.10. ábra. *A tanulási ráta adaptív változtatása.*

számú lépés óta nem tudta a korábbi legjobb eredményét meghaladni. Szintén hatásos módszer a koszinuszos lágyítás alkalmazása, ami a tanulási rátát egy maximum és minimum érték között egy koszinusz függvény első fél periódusának megfelelően állítja. A koszinusz lágyítás alkalmazásakor a félperiódus befejezésekor tovább lehet folytatni a tanítást a maximális tanulási értéket használva és újabb lágyítást végezni. Ennek értelme, hogy a hirtelen megnövelt tanulási ráta „kilöki" a háló súlyait a lokális minimumból és a további tanulás során egy közeli jobb lokális minimum megtalálását teszi lehetővé.

10.5. Adatbázisok előállítása

A neurális hálók tanításának harmadik nehézsége a nagy számú címkézett tanítóadat előállítása. Ez a probléma kis mértékben csökkenthető a már korábban ismertetett adat augmentáció módszerével. További említésre méltó módszer az úgynevezett félig felügyelt tanulás, amikor az adatbázisnak csak egy kis részhalmaza címkézett, a maradék adat esetén azt várjuk el a hálótól, hogy a hasonló adatok hasonló címkét kapjanak.

10.5.1. Transfer learning

A mély tanulás területének egyik legjelentősebb áttörése ezt a problémát orvosolja. Beláttuk ugyanis, hogy a konvolúciós neurális hálók első konvolúciós és leskálázó rétegekből álló része különböző képi jellemzők detektálását végzi el. Ahogy előrefele haladunk a háló rétegei között úgy

ezek a jellemzők egyre komplexebbé, egyre feladatspecifikusabbá válnak. Ebből következik azonban, hogy ha már van egy valamilyen feladatra betanított hálózatunk, akkor annak elülső rétegei felhasználhatók egy másik, hasonló feladat elvégzésére. Így elegendő az új feladathoz csak a háló utolsó rétegeit újra tanítani, amihez lényegesen kevesebb adat elegendő, hiszen kevesebb szabad paramétert tartalmaznak, mint az egész háló.

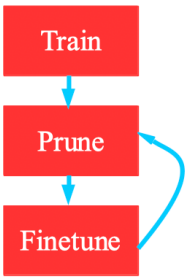
Ezt a technikát transzfer tanulásnak nevezzük, és elterjedt megoldás a mély tanulás területén. A fent ismertetett érvelés annyira igaz, hogy számos esetben elegendő a hálók legutolsó, lineáris rétegét újra tanítani. Más esetekben szükség lehet az elülső hálórészek finomhangolására, azonban ehhez is nagyságrendekkel kevesebb adat elegendő lehet.

10.6. Installáció

A jelen fejezet utolsó témája a neurális hálók gyakorlati felhasználásra való felkészítésének (installálásának) kérdései. Neurális hálózatok telepítése esetén két probléma adódik: a neurális hálók ugyanis több millió paraméterrel rendelkeznek, vagyis egy mély háló mérete meglehetősen nagy. Ráadásul végrehajtásuk több milliárd műveletet igényel, így meglehetősen lassúak is. Ez nagymértékben megnehezíti a kisebb teljesítményű eszközökben való használatukat.

10.6.1. Pruning

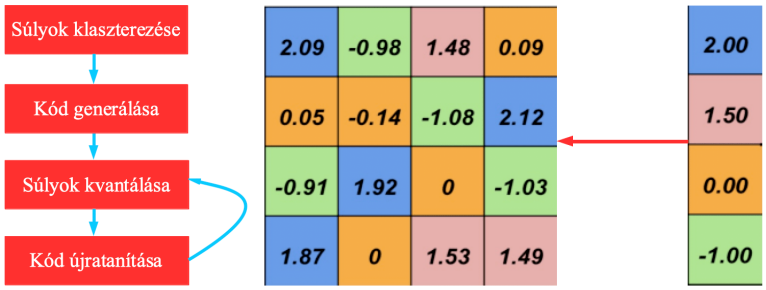
A fenti problémákra adott megoldások közül az egyik legfontosabb a tisztítás művelete (pruning), amely során a háló súlyai közül kiválasztjuk a legkevésbé fontos néhány százalékot és ezeket töröljük. A súlyok rangsorolására számos módszer létezik, amelyek közül a legnyilvánvalóbb egyszerűen a súlyok abszolút értékének használata. Ezt követően a törölt súlyok nulla értéken tartása mellett finomhangoljuk a hálót, majd ezt a két lépést többször megismételjük. Több kutatás is alátámasztja, hogy az iteratív tisztítás technikájával a háló súlyainak 90%-át törölni lehet csupán néhány százalékos teljesítménybeli veszteség mellett. Ez nyilvánvalóan tízszeres gyorsulást eredményez.



10.11. ábra. *A Pruning módszere.*

10.6.2. Weight sharing

Hasonlóan hatékony módszer a súlyok kvantálása, melynek lényege, hogy a súlyokat egy klaszterező algoritmus segítségével néhány csoportba szedjük, majd minden súlyt a klaszterek középértékével helyettesítünk. Ezt követően a klaszterközéppontok értékét finomhangoljuk, és a tisztításhoz hasonlóan ezeket a műveleteket is iteratívan végezzük. Egy átlagos neurális háló súlyait ilyen módon be lehet osztani 16 csoportba csupán néhány százalékos teljesítményvesztés mellett. Mivel azonban csak 16 különböző fajta súly van a hálóban, ezért egy súlyt elegendő 4 bit felhasználásával ábrázolni. Ez a szokásos 32 bites lebegőpontos számábrázoláshoz képest nyolcszoros tömörítést jelent.



10.12. ábra. *A Weight sharing módszere (bal) és a súlyok kvantálásának sémája (jobb).*

10.6.3. Ensemble

Egy érdekes eljárás még a modell együttesek (ensemble) használata. Ebben az esetben általában több különböző felépítésű, és eltérő módon inicializált és tanított hálózat kimenetének összeátlagolásával állítunk elő egy ”konszenzus” becslést, amely a tapasztalatok alapján a legjobb háló kimenetét hozzávetőlegesen 2-3%-kal képes javítani. Az ilyen jellegű módszereket gyakran hívják szakértő rendszereknek is.

További Olvasnivaló

[18] J. Heaton, “Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning”, *Genetic Programming and Evolvable Machines*, 19. évf., 1-2. sz., 305–307. old., 2017. okt. DOI: 10.1007/s10710-017-9314-z. cím: https://doi.org/10.1007%2Fs10710-017-9314-z.

[26] K. He, X. Zhang, S. Ren és J. Sun, “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”, *2015 IEEE International Conference on Computer Vision (ICCV)*, IEEE, 2015. dec. DOI: 10.1109/iccv.2015.123. cím: https://doi.org/10.1109%2Ficcv.2015.123.

[27] S. Ioffe és C. Szegedy, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, 2015. eprint: 1502.03167. cím: http://www.arxiv.org/abs/1502.03167.

11. fejezet

Detektálás és szegmentálás

11.1. Bevezetés

A tárgy bevezető előadásában a számítógépes látás több fontos feladatát is felsoroltuk, egyelőre azonban csak az osztályozás megvalósítását ismertettük. A jelenlegi előadásban az objektumdetektálás és a szegmentáció különböző fajtáit vizsgáljuk.

11.2. Szemantikus szegmentálás

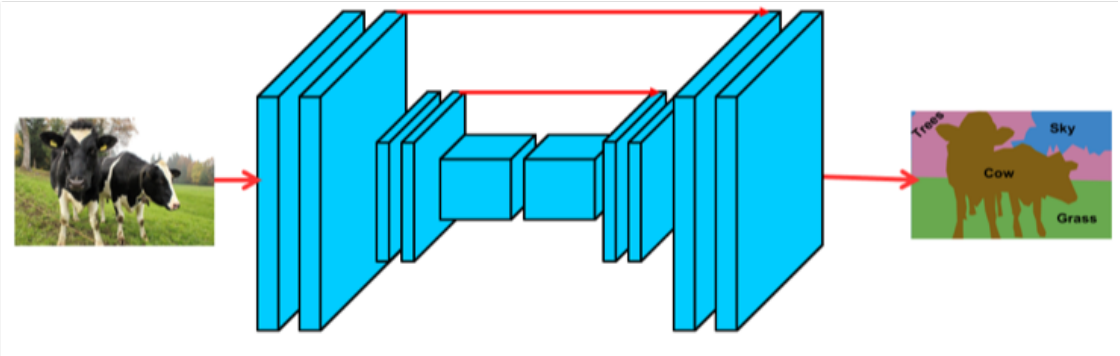
Az osztályozáshoz a legközelebb álló feladat a szemantikus szegmentálás, melynek során a kép összes pixelét kívánjuk osztályozni. Ez természetesen egy osztályozó neurális háló felhasználásával egy csúszóablakos eljárással elvégezhető, azonban ezt egy átlagos kép százezres, vagy milliós nagyságrendben lévő összes pixelére elvégezni rendkívül hosszú ideig tartana.

11.1. ábra. *A szemantikus szegmentálás feladata.*

11.2.1. Teljesen konvolúciós architektúra

Éppen ezért célszerű lenne a folyamatot párhuzamosítani úgy, hogy egyetlen neurális háló segítségével elvégezhető legyen. Erre a teljesen konvolúciós hálózatok (FCN – Fully Convolutional Network) alkalmasak melyek egyszerűen konvolúciós és aktivációs rétegek sorából állnak. A háló utolsó rétege is konvolúciós, kimeneti csatornáinak száma az osztályok számával egyezik meg, így a kimeneti aktivációs térkép elemei az egyes pixelek osztályozásának tekinthetők. Az architektúra problémája, hogy a kép teljes eredeti felbontásán elvégzett konvolúciók drágák, így érdemes leskálázó operációkat is beiktatni. Ekkor viszont a kimeneti osztályozás is kisebb felbontású lesz, ami nem kívánatos.

A gyakorlatban használt FCN hálók egy fel- és leskálázó részből állnak, melyek többé-kevésbé egymás tükörképei. Ily módon az eredeti kép felbontásával megegyező méretű kimenetet kapunk, de a feldolgozás zömét alacsonyabb felbontáson végezzük, így a futási idő is elfogadható lesz. Érdemes még megjegyezni, hogy az FCN hálók általában tartalmaznak az azonos felbontású fel- és leskálázó részek között rövidzár összeköttetéseket, ami segíti a gradiensek áramlását, így a tanítás konvergenciáját. Az összeköttetések másik előnye, hogy a háló elején detektált alacsony szintű jellemzők segítenek az osztályok határvonalának minél pontosabb meghatározásában, amik a leskálázás során elvesznek.



11.2. ábra. *Egy tipikus FCN architektúra.*

Az FCN hálók teljesítménye számos további módszerrel javítható. Ezek közül az egyik legegyszerűbb a reziduális, vagy dense blokkok használata a háló leskálázó részében. Ezek használatával elérhető, hogy mély, nagy effektív látómezővel rendelkező hálót használjunk szegmentálásra a konvergencia megnehezítése nélkül.

További lehetőség a dilatált (angol irodalomban néha atrous néven említett) konvolúciós szűrők használata. A dilatáció hatása, hogy a szűrő effektív látómezejét megnöveli azonos paraméterszám mellett. Így ugyanaz a háló nagyobb kontextust képes vizsgálni, így javítható a szegmentálás konzisztenciája. Szintén hasonló javítást lehet elérni a térbeli piramis pooling (Spatial Pyramid Pooling) használatával. Ez a módszer a háló végén előálló aktivációs térképen több, különböző méretű pooling operációt végez el párhuzamosan, az így keletkező aktivációkat pedig konkatenálja, az osztályozást pedig az így keletkező jellemzők segítségével végzi el. Ennek a megoldásnak az előnye, hogy a több skálafaktor mellett előálló aktivációk együttes használatával a háló skálaérzékenységét csökkentjük. A piramis pooling műveletét is szokás dilatált módon végezni hasonló megfontolásokból.

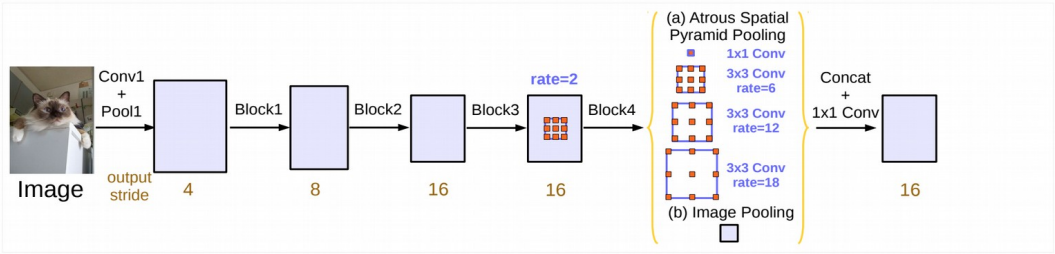
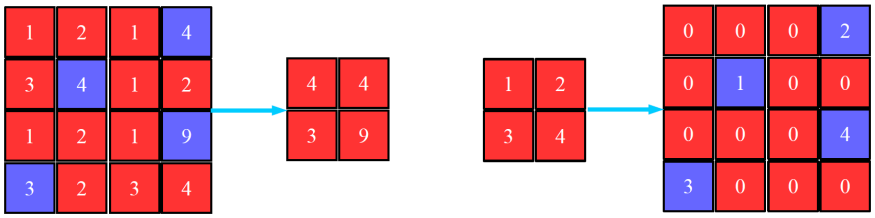


Figure 5. Parallel modules with atrous convolution (ASPP), augmented with image-level features.

11.3. ábra. *Egy dilatált piramis poolingot használó architektúra.*

11.2.2. Felskálázás módszerei

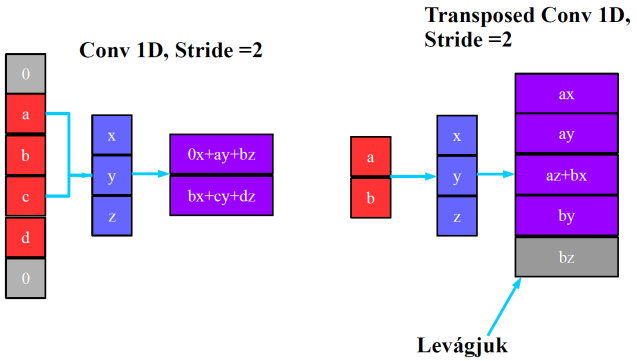
Az egyetlen megmaradó kérdés azonban az, hogy hogyan lehetséges a felskálázást konvolúciós neurális hálózatokban megvalósítani. Erre az egyik legegyszerűbb ötlet az úgynevezett unpooling operáció, melynek működése annyiból áll, hogy a leskálázó részben elvégzett maximum pooling



11.4. ábra. *A max unpooling művelet.*

során eltároljuk, hogy melyik pixel pozícióban volt a maximum érték, a felskálázás során pedig ebbe a pozícióba írjuk az alacsonyabb szint értékét, míg a többi pozíció nulla marad.

Szintén elterjedt megoldás a transzponált konvolúció, amely tulajdonképpen a stride-dal végzett konvolúció megfordítása. A módszer elnevezése onnan ered, hogy a konvolúció leírható, mint egy mátrixszal való szorzás, a transzponált konvolúció pedig ennek a mátrixnak a transzponáltjával történő szorzás. Ennek a módszernek nagy előnye, hogy a felskálázás tanulható, amely nagymértékben javítja a szegmentálás minőségét.



11.5. ábra. *A traszponált konvolúció 1D-ben.*

A traszponált konvolúció neve onnan ered, hogy a konvolúció leírható egy mátrixszorzással az alábbi módon:

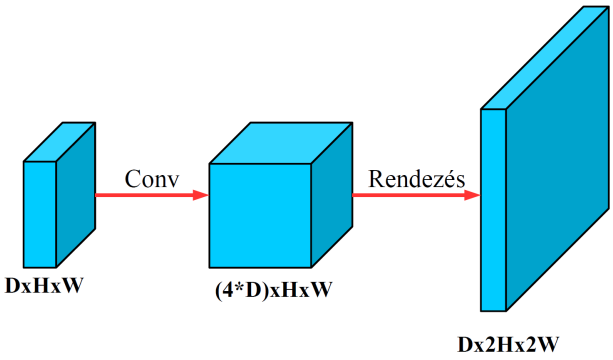
$$\begin{pmatrix} x & y & z & 0 & 0 & 0 \\ 0 & x & y & z & 0 & 0 \\ 0 & 0 & x & y & z & 0 \\ 0 & 0 & 0 & x & y & z \end{pmatrix} \begin{pmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{pmatrix} = \begin{pmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + by \end{pmatrix} = Xa \tag{11.1}$$

A traszponált konvolúció akkor a korábbi mátrix transzponáltjával történő szorzásnak felel meg. Érdemes megjegyezni, hogy a mély tanulás területén a transzponált konvolúciót gyakorta szokás - helytelenül - dekonvolúciónak hívni. Ez a tévedés egy mátrix transzponáltjának és inverzének összekeverésével ekvivalens.

$$\begin{pmatrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} ax \\ ay + bx \\ az + by + cx \\ bz + cy + dx \\ cz + dy \\ dz \end{pmatrix} = X^T a \tag{11.2}$$

Létezik még egy harmadik elterjedt módszer is, ez a sűrű felskálázó konvolúció. Ennek lényege, hogy egy konvolúciós réteg segítségével az adott aktivációs térkép csatornáinak számát a négyszeresére növeljük, majd az így kapott tömböt átrendezzük úgy, hogy az aktivációs térkép térbeli

méretei az eredeti kétszeresei legyenek, csatornáik száma pedig egyezzen meg az eredetivel. Ez a módszer szintén tanuló felskálázás, viszont több paraméterrel rendelkezik, így képes komplexebb transzformációk megtanulására lassabb működés árán.



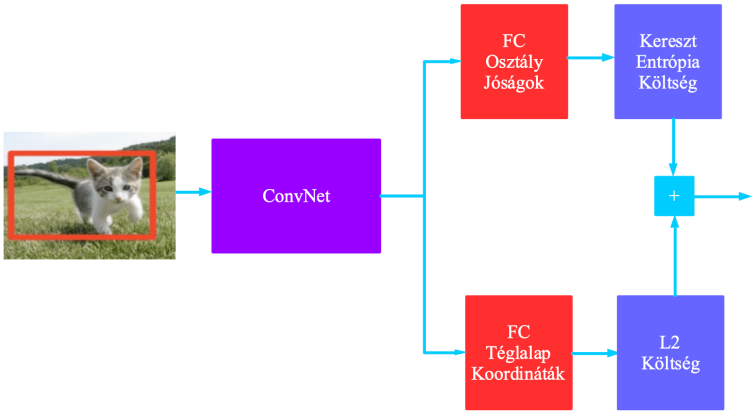
11.6. ábra. *A sűrű felskálázó konvolúció (DUC).*

11.3. Detektálás

A szemantikus szegmentáció tárgyalásának befejeztével a másik fontos tématerülettel, a detektálással foglalkozunk. Ennek során valamivel egyszerűbb a kinyert jellemző (általában befoglaló téglalapok), azonban megoldható az egyes objektumok különválasztása is.

11.3.1. Lokalizáció

Ennek az egyik legegyszerűbb változata a lokalizáció, amikor az osztályozás feladatát az adott osztályú objektum befoglaló téglalapjának meghatározásával egészítjük ki. Ez a feladat szintén könnyedén elvégezhető neurális hálók segítségével, hiszen nincs más dolgunk, mint egy osztályozó hálóhoz újabb négy kimenetet hozzáadni. Ezekre a kimenetekre előírjuk, hogy a befoglaló téglalap négy paraméterét minél pontosabban adja ki a neurális háló. Mivel ez egy regressziós probléma, ezért a téglalap paramétereinek pontosságát a négyzetes hiba költségfüggvénnyel célszerű mérni. A lokalizációs háló teljes hibája az osztályozás és a regresszió hibafüggvényeinek összege lesz.



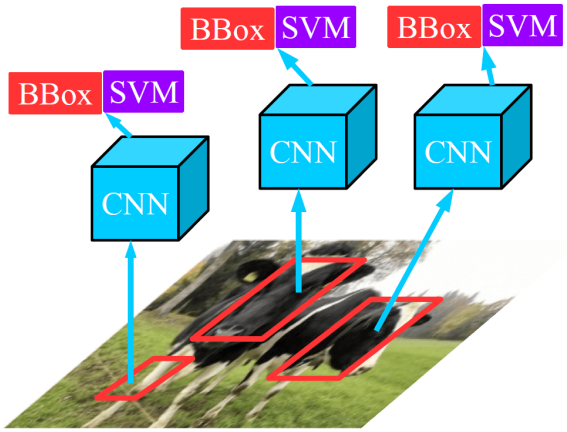
11.7. ábra. *A lokalizációt megvalósító architektúra.*

11.3.2. Régió-CNN

A detektálás feladata esetén azonban már lényegesen nehezebb dolgunk van. Ennek oka, hogy akárhány, akármilyen osztályú objektum előfordulhat, az architektúrát pedig ennek fényében kell

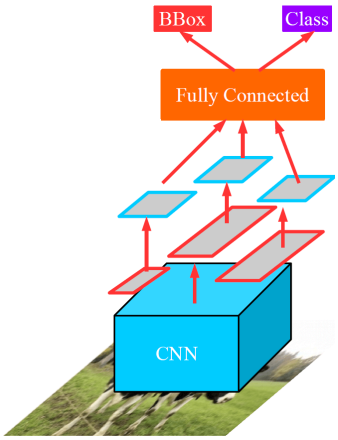
megalkotnunk. Természetesen az objektumok számára egy durva felső becslést adhatunk, így készíthetnénk egy olyan konvolúciós hálót, aminek pontosan ennyi különböző osztályozó és téglalap becslő kimenete van. Ez azonban N maximális objektum és C osztály esetében $N*(C+4)$ kimenetet jelentene, ami rengeteg lehet tekintve, hogy N a néhány tucat, C pedig a százas, vagy az ezres nagyságrendben mozoghat.

Egy alternatív megoldást jelenthet, ha felhasználjuk a korábbi előadások során tárgyalt régiójavasló módszereket. Ezek az eljárások tulajdonképpen hagyományos szegmentálási módszerek, amelyek segítségével összefüggő régiójavaslatokat állíthatunk elő. Ezeken a régiójavaslatokon ezt követően egy lokalizációra használható konvolúciós neurális hálózatot futtatunk le egyesével. Ezt a módszert R-CNN (Region Convolutional Neural Net) néven ismerjük. Érdemes megjegyezni, hogy a felhasznált lokalizációs háló osztályozó kimenetének a releváns osztályokon felül tartalmaznia kell egy „egyik sem” kimenetet az objektumokat nem tartalmazó régiójavaslatok kiszűréséhez.



11.8. ábra. *A R-CNN architektúra.*

Az R-CNN módszer egyik legfontosabb hátránya, hogy az összes régiójavaslaton külön-külön futtatjuk le a neurális hálót, ami pazarlás. A módszer egy továbbfejlesztése, a FastR-CNN az egész képen futtat egy csak konvolúciós és leskálázó rétegekből álló hálót, majd az ez által elkészített aktivációs térképen keres régiójavaslatokat. Ezt követően ezeket a javaslatokat egy speciális pooling művelet segítségével azonos méretűre hozza (a hagyományos pooling műveletek egy adott faktorral skáláznak). Ezt követően a régió javaslatokon már csak egy kis méretű, csak lineáris rétegekből álló hálót futtat, amelyek az osztály és a téglalap becslését végzik (11.8. ábra). Ez a módszer az eredeti R-CNN módszerhez képest 10-20-szor gyorsabban működik.



11.9. ábra. *A Fast R-CNN architektúra.*

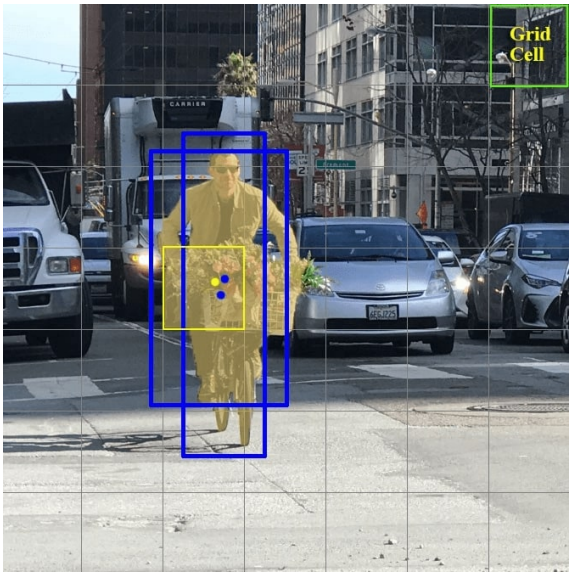
A FastR-CNN működésének a leglassabb része a régiójavaslatok előállítása, ami a futási idő 90%-át teszi ki. Éppen ezért megalkottak még egy továbbfejlesztett változatot, ami a régiójavaslatok előállítását is egy RPN (Region Proposal Net) nevű neurális háló segítségével végzi el. Ez a háló

a kezdeti konvolúciós rész által előállított aktivációs térképből állít elő fix számú régió javaslatot, amelyek mindegyikét binárisan osztályozza (objektum/nem objektum). Erre azért van szükség, mert a fix számú régió kimenet miatt a háló fixen ennyi régiójavaslatot tesz. A fő detektáló háló tanítása mellett az RPN hálót is arra tanítjuk, hogy a régiók befoglaló téglalapját és „objektumszerűségét" minél nagyobb pontossággal találja el. Ez a módszer a FastR-CNN megoldáshoz képest egy újabb tízszeres gyorsítást eredményez.

11.3.3. YOLO

Fontos azonban tudni, hogy nem csak régiójavaslatok segítségével lehet hatékony objektumdetektálást végezni. Erre kitűnő példa a rendkívül népszerű YOLO (You Only Look Once) architektúra, mely nem összetévesztendő a megegyező rövidítésű szállóigével. A YOLO megoldás alapvetően hasonlít a detektálás tárgyalásának elején felvetett sok külön lokalizáló kimenetet javasló megoldáshoz. A működése során a YOLO a képet először egy tisztán konvolúciókból és leskálázásból álló hálón küldi végig, így előállítva a végső becslésekhez felhasznált aktivációs térképet.

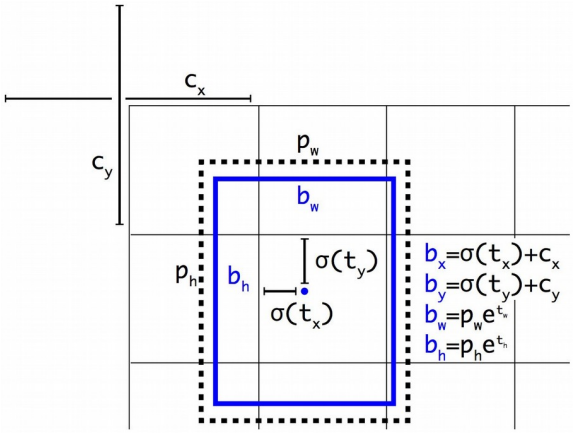
A végső becsléshez a YOLO a képet egy NxN-es rács segítségével felosztja, és minden cellából B darab objektumjelölt téglalapot becsül. Minden téglalaphoz tartozik egy C kimenetű osztályozó, valamint bináris osztályozó is, amely az adott téglalapba eső képrészlet „objektumszerűségét" adja meg. Így minden egyes cella esetén Bx(5+C) kimenete van a hálónak, amelyet egy 1x1 méretű konvolúciós szűrővel állít elő. Érdemes megjegyezni, hogy a téglalapok pozícióját a YOLO cella bal felső sarkához képest becsüli meg, így minden objektum detektálásáért az a cella felelős, amelyikben az objektum középpontja található.



11.10. ábra. *A YOLO modell által készített rács és becslések.*

A befoglaló téglalap szélességét és magasságát a YOLO egy referencia téglalaphoz (ún. anchor box) képest becsli meg, amelyből összesen B darab van (minden becsléshez egy). Az egyes anchor boxok szélesség és magasság értékeit a tanító adatbázisban szereplő téglalapokon végzett B elemű klaszterezés segítségével határozzuk meg. Fontos még említeni, hogy a detektálás során a YOLO egy objektumot többször is megtalálhat, mely esetben a túlságosan hasonló alakú predikciók közül a legnagyobb konfidencia értékűt tartjuk meg, míg a többit eldobjuk. Ezt a lépést nevezzük non-maximum suppression-nek.

A YOLO-nak több változata is létezik, az anchor boxok használatát például a második verzió vezette be. A harmadik verzió a detektálást három különböző skálafaktor mellett is elvégzi, amihez a szegmentálás során megismert felskálázás és előrecsatolás trükkjeit alkalmazza. Ezzel a megoldással a YOLO teljesítménye jelentősen javul kis méretű objektumok pontos detekciója esetén. A YOLO legfőbb erénye a régió alapú detektálással szemben, hogy rendkívül gyors, így megfelelő

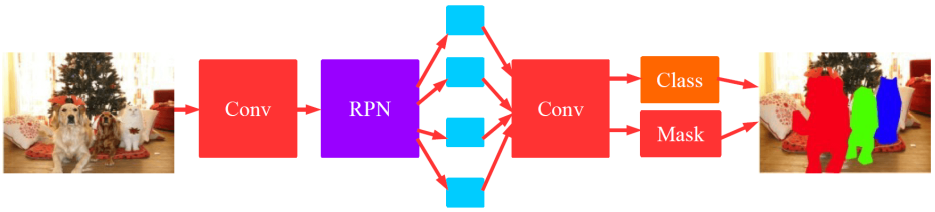


11.11. ábra. *A YOLO téglalap becslési módja. A téglalap koordinátáit a rács bal felső sarkához képest, a méreteit pedig az anchor box-hoz viszonyítva becsüljük.*

hardver használata esetén valósidejű működésre is képes, különösképp a TinyYOLO névre hallgató változata.

11.3.4. Mask-RCNN

Az alfejezet végén érdemes még említést tenni a jelenlegi témakör utolsó feladatáról, amely az objektumszegmentálás volt. Ebben az esetben nem csupán szemantikus módon kívánjuk szegmentálni a képet, hanem az azonos osztályba tartozó egyes objektumokat is szeretnénk megkülönböztetni. Bár ez nyilvánvalóan a legnehezebb feladat az összes közül, az eddigi ismeretek alapján egy ilyen architektúra mégis könnyedén megérthető. Az RPN alapú objektumdetektálás során az egyes objektumokat tartalmazó képrészleteket ugyanis már előállítottuk, így a feladatunk csak annyiban különbözik, hogy a befoglaló téglalap helyett minden objektumhoz egy bináris maszkot kell előállítanunk. Ezt könnyedén megtehetjük a szemantikus szegmentálásból ismert felskálázó hálórész segítségével. Ezt az architektúrát Maszk R-CNN néven ismerik.



11.12. ábra. *A Mask-RCNN architektúra.*

További Olvasnivaló

[18] J. Heaton, “Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning”, *Genetic Programming and Evolvable Machines*, 19. évf., 1-2. sz., 305–307. old., 2017. okt. DOI: 10.1007/s10710-017-9314-z. cím: https://doi.org/10.1007%2Fs10710-017-9314-z.

[28] J. Long, E. Shelhamer és T. Darrell, “Fully convolutional networks for semantic segmentation”, *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2015. jún. DOI: 10.1109/cvpr.2015.7298965. cím: https://doi.org/10.1109%2Fcvpr.2015.7298965.

[29] S. Ren, K. He, R. Girshick és J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39. évf., 6. sz., 1137–1149. old., 2017. jún. DOI: 10.1109/tpami.2016.2577031. cím: https://doi.org/10.1109%2Ftpami.2016.2577031.

[30] J. Redmon, S. Divvala, R. Girshick és A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection", *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2016. jún. DOI: 10.1109/cvpr.2016.91. cím: https://doi.org/10.1109%2Fcvpr.2016.91.

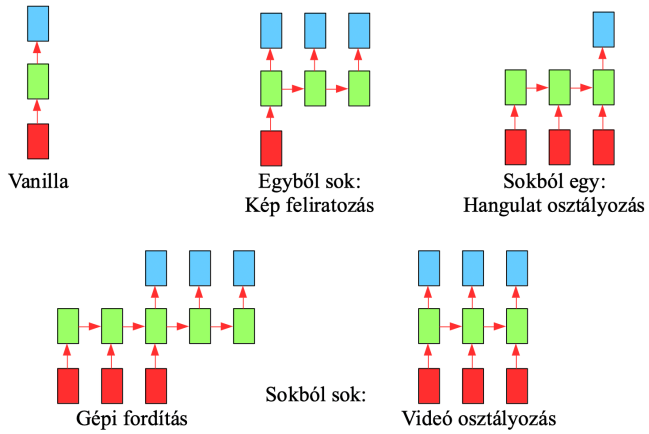
[31] K. He, G. Gkioxari, P. Dollar és R. Girshick, "Mask R-CNN", *2017 IEEE International Conference on Computer Vision (ICCV)*, IEEE, 2017. okt. DOI: 10.1109/iccv.2017.322. cím: https://doi.org/10.1109%2Ficcv.2017.322.

12. fejezet

Visszacsatolt hálózatok

12.1. Bevezetés

Ez eddigi diszkusszió során olyan módszereket ismertünk meg, amelyek állóképek egymástól független feldolgozására alkalmasak. Képsorozatok feldolgozásának azonban számos jelentős alkalmazása van, többek között a videók osztályozása, vagyis más néven az eseménydetektálás. Könnyen belátható, hogy ahogy bizonyos alkalmazások esetében szükség lehet a képen található objektumokat azonosítani, úgy még hasznosabb lehet egy videón lejátszódó eseményt vagy cselekményt felismerni. Egy valamelyest eltérő alkalmazás képek feliratozása, melynek során egy képhez nem egyetlen címkét, hanem egy egész mondatot rendelünk, így lényegesen komplexebb leírást tudunk adni. Ebben az esetben nem a háló bemenete, hanem annak kimenete értelmezhető sorozatként.



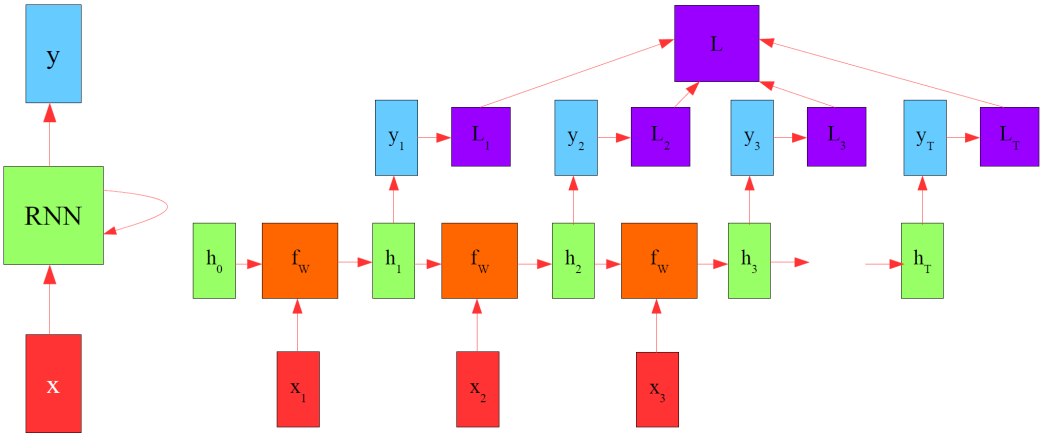
12.1. ábra. *Különböző sorozatfeldolgozási feladatok.*

12.2. Visszacsatolt neurális hálók

Könnyen belátható azonban, hogy az előrecsatolt konvolúciós hálóknak memória eleme nincs, így nem igazán alkalmas időbeli sorozatok feldolgozására. Sorozatok hatékony feldolgozásához viszont egy olyan új háló struktúrára lesz szükségünk, amely valamilyen belső állapottal is rendelkezik. Az ilyen hálózatokat visszacsatolt neurális hálózatoknak (RNN – Recurrent Neural Network) nevezzük. A visszacsatolt réteg működése során a belső állapotának aktuális értékét az aktuális bemenet és az egy lépéssel korábbi belső állapot értéke alapján számolja ki. A cella kimenete pedig a belső állapot imént kiszámolt aktuális értékétől függ. Egy RNN cella egyenlete az alábbi módon adódik:

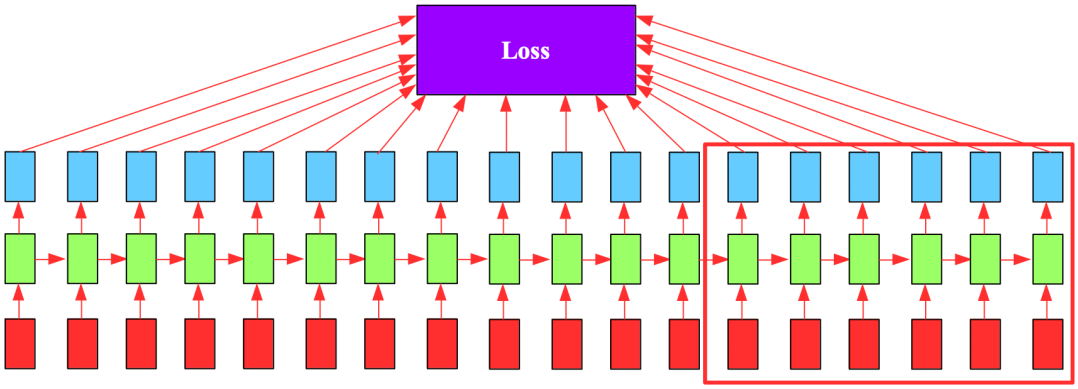
$$\begin{aligned} h_t &= tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ y_t &= \sigma(W_{hy}h_t) \end{aligned} \tag{12.1}$$

Ahol h a belső állapotot jelöli, t pedig az aktuális időpillanat. Könnyen belátható, hogy egy RNN cella tulajdonképpen három lineáris réteg és egy aktivációs függvény együtteseként adódik. Az új architektúra bevezetésével azonban felmerül az a kérdés, hogy hogyan lehet ebben a struktúrában a súlyok gradienseit meghatározni. Probléma ugyanis, hogy a backpropagation módszere visszacsatolt architektúrák esetén nem működik. Ez szerencsére azonban egy egyszerű trükkel orvosolható: egy visszacsatolt háló ugyanis átalakítható egy hagyományos előrecsatolt hálóvá az időben történő kibontás műveletével. Ez azt jelenti, hogy az egyetlen RNN réteg különböző időpontokban felvett állapotára úgy tekintünk, mint egy hagyományos háló egymást követő rétegeire.



12.2. ábra. *Egy RNN cella felépítése (bal) és időbeli kibontása (jobb).*

Mivel egy RNN cellának minden időpontban van kimenete és hibája, ezért a kibontott háló minden rétegéhez fog tartozni egy-egy kimenet és hiba, melyeknek összege adja ki a teljes hibát. Innentől a már megismert backpropagation algoritmus minden további nélkül használható. Két fontos különbség adódik azonban a hagyományos előrecsatolt hálókhoz képest. Egyrészt, ahogy haladunk előre az időben a kibontott háló mérete egyre növekszik, ami a tanítás folyamatának lassulásával jár. Ráadásul a helyes működéshez és tanításhoz nem szükséges a végtelenségig emlékezni a múltbeli bemenetekre. Éppen ezért a kibontás során a háló maximális méretét korlátozzuk és a legrégebbi réteget és bemenetet töröljük a kibontott hálóból.

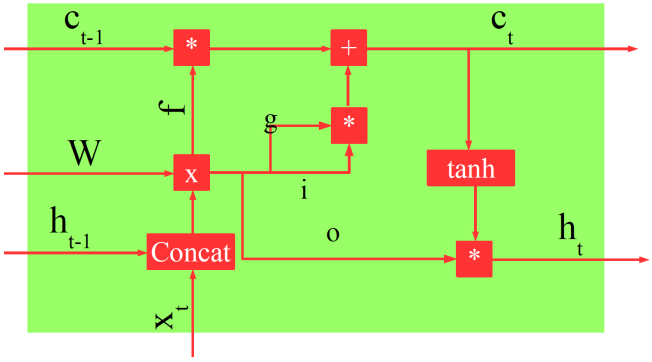


12.3. ábra. *A Backpropagation through time (BPTT) algoritmus elve. Amikor a háló végén járunk, csak a piros kerettel jelölt részig végezzük el a visszaterjesztést, különben a probléma mérete a végtelenségig növekedne.*

12.2.1. LSTM

A másik fontos különbség, hogy a kibontott sokrétegű háló esetén minden réteg súlymátrixa azonos (hiszen valójában egyetlen visszacsatolt rétegről van szó). Ez azt jelenti, hogy amikor a láncszabály segítségével a deriváltakat előállítjuk, akkor az egyes rétegek deriváltjainak egy hosszú szorzatát

kell kiszámolnunk. Ebben az esetben azonban a szorzat minden eleme azonos, vagyis valójában egy hatványról beszélhetünk. Azt pedig könnyen beláthatjuk, hogy a gyakorlatban bármilyen szám vagy mátrix sokadik hatványa vagy nulla vagy végtelen, kivéve, ha az a szám pontosan 1. Ebből következik, hogy egy visszacsatolt cella gradiensei könnyedén eltűnnek, vagy „felrobbannak”, ami a tanítást ellehetetleníti.



12.4. ábra. *Az LSTM cella felépítése.*

Erre a problémára az egyetlen lehetséges megoldás az, ha olyan struktúrát alkotunk, ahol a belső állapot aktuális és egy lépéssel korábbi állapota közti derivált nagyjából egy. Pontosan ilyen architektúra az LTSM (Long Short-Term Memory) cella. A cella elnevezése onnan ered, hogy a készítői egy olyan rövidtávú memóriacellát kívántak alkotni, amely a gyakorlati használhatósághoz megfelelően hosszú ideig képes emlékezni az RNN cellával szemben. Míg az RNN cella három lineáris egységből állt, az LSTM négy aktivációs függvényt is tartalmazó egységből áll, melyeket kapuknak nevezünk.

Az LTSM cella működése során az egyes kapuk hatása nélkül a c cella állapot korábbi értéke változás nélkül átíródik az aktuális állapotba, így a kettő közötti derivált pontosan egy. A cella állapot értéke azonban még az egyes kapuk hatására módosulhat. Az első ilyen kapu a felejtés kapu f, amely egy a cella állapottal azonos méretű vektor, melynek minden eleme nulla és egy között van a szigmoid nemlinearitás hatására. A cella állapot vektorát ezzel a vektorral elemenként megszorozva a cella állapot bizonyos részleteit elfelejtjük.

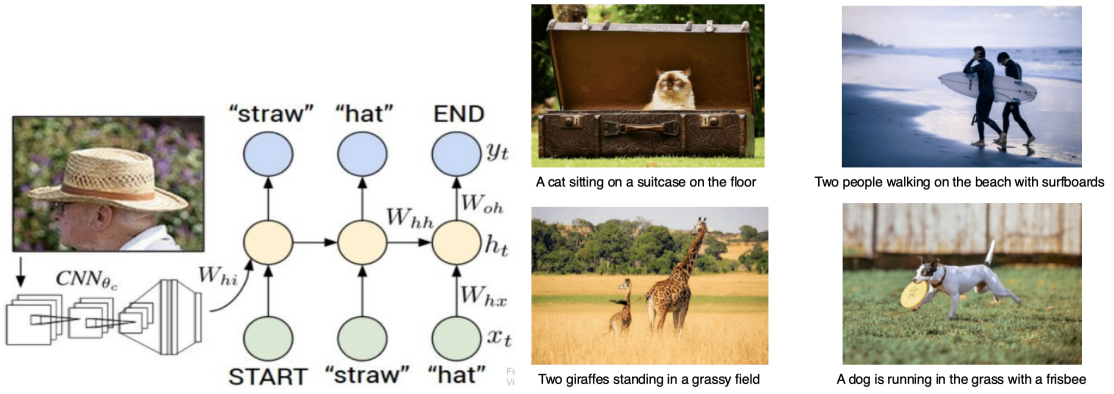
A következő kapu az úgynevezett főkapu g, amelynek feladata, hogy a bemenet aktuális értékéből és a cella kimenet korábbi értékéből kinyerje azokat a jellemzőket, amelyek a cella állapotban megjegyezhetők. Ezt követően a felejtés kapuval analóg i bemeneti kapu vektorával a főkapu vektorát elemenként szorozzuk, ezáltal kiválasztva a megjegyezhető jellemzőkből a releváns részeket, majd ezt a cella állapothoz hozzáadjuk. A végső lépés a cella aktuális kimenetének előállítása, amire a cella állapotnak az o kimeneti kapu által szűrt értékeit adjuk ki.

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \tag{12.2}$$
$$c_t = f * c_{t-1} + i * g$$
$$h_t = o * tanh(c_t)$$

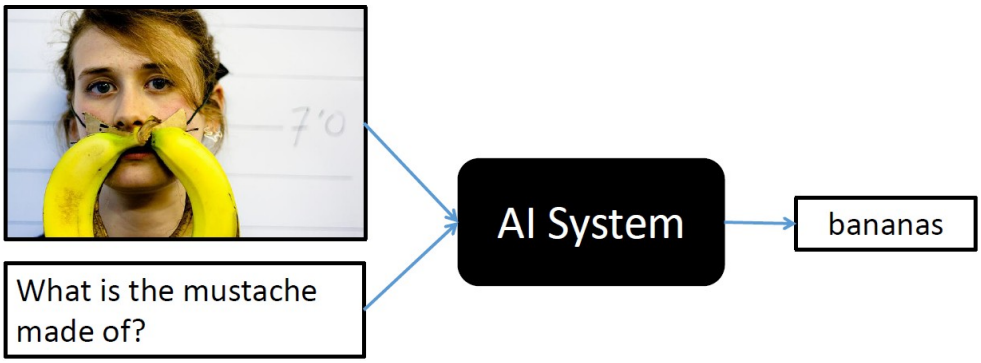
Ahol $*$ az elemenkénti szorzást jelöli. Érdemes megjegyezni, hogy az LTSM cellának számos apróságokban eltérő variációja létezik, valamint léteznek nagyobb eltérést mutató, de hasonló alapötletre épülő visszacsatolt cellák. Ilyen például az úgynevezett kapuzott visszacsatolt cella (GRU – Gated Recurrent Unit). Szintén érdemes észrevenni, hogy a gradiensek minél zavartalanabb hátrafele történő áramlásának elősegítése úgynevezett „rövidzár” kapcsolatok segítségével nem itt fordult elő először. Az előző fejezetben ismertetett reziduális blokk alapötlete ehhez rendkívül hasonló volt.

12.2.2. Alkalmazási példák

A videók osztályozásán felül számos további érdekes alkalmazása van a sorozatokon végzett feldolgozási módszereknek. Ezek közül kiemelendő a képek feliratozása, melynek során a bemenetként kapott állóképekhez egy rövid, 1-2 mondatos leírást rendelünk. Ebben az esetben a visszacsatolás a mondatok generálásánál jelentkezik. További fontos alkalmazás a különböző (vizuális) kérdés-válasz rendszerek megvalósítása. Itt az algoritmusnak egy bemeneti képről vagy videóról feltett kérdésre kell válaszolnia (Pl.: "Milyen színű sapka van a napszemüveges férfin?"). Érdemes még megemlíteni a különböző explicit memóriával rendelkező rendszerek (pl. Turing-gép) implementációját is.



12.5. ábra. *A képek feliratozásának elve (bal) és eredménye(jobb).*



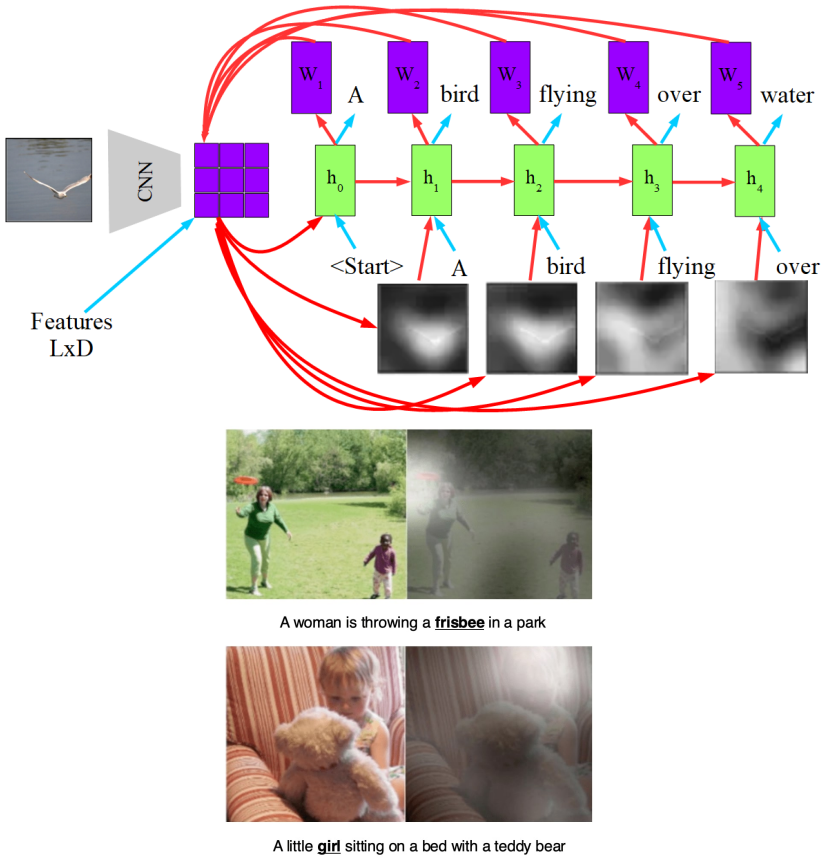
12.6. ábra. *A vizuális kérdés-válasz rendszer problémája.*

A visszacsatolt hálózatok egy rendkívül érdekes alkalmazása az úgynevezett puha figyelem modell megvalósítása. Ennek során egy visszacsatolt cella a kép tartalma alapján a kép egyes részeihez kiad egy-egy súlyt és az adott lépésben az egyes részeket ezekkel a súlyokkal veszi figyelembe. A működés során a cella minden egyes lépésben új súlyokat generál, így folyamatosan változik, hogy a hálózat a kép melyik területeire összpontosít. Képek feliratozásakor megfigyelhető, hogy a mondat generálása során az egyes szavak kiadásának pillanatában a háló a képen pont oda figyel, ahol az adott szónak megfelelő tárgy található.

További Olvasnivaló

[18] J. Heaton, "Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning", *Genetic Programming and Evolvable Machines*, 19. évf., 1-2. sz., 305–307. old., 2017. okt. DOI: 10.1007/s10710-017-9314-z. cím: https://doi.org/10.1007%2Fs10710-017-9314-z.

[32] S. Hochreiter és J. Schmidhuber, "Long Short-Term Memory", *Neural Computation*, 9. évf., 8. sz., 1735–1780. old., 1997. nov. DOI: 10.1162/neco.1997.9.8.1735. cím: https://doi.org/10.1162%2Fneco.1997.9.8.1735.



12.7. ábra. *A puha figyelem elve (fent) és hatása a feliratozásra (lent). Ezeken a képeken a háló által generált figyelem súlyok látszanak az aláhúzott szó kiadásának időpillanatában*

[33] K. Xu, J. Ba, R. Kiros és tsai., *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*, 2016. eprint: `1502.03044`. cím: `http://www.arxiv.org/abs/1502.03044`.

III. rész

3D Látás

13. fejezet

Kameramodell és Kalibráció

13.1. Bevezetés

A számítógépes látás alapvető célja, hogy egy kamera képe(i) alapján a valós világról automatikusan információkat tudjunk nyerni egy számítógépes algoritmus segítségével. A legtöbb gyakorlatban használt képalkotó rendszer azonban a valós, háromdimenziós világról egy kétdimenziós vetületet készít, amely során számos információ teljes mértékben elveszik vagy torzul. Egy tipikus képen nem maradnak meg az egyes képpontok kamerától mért távolságai, így ezeket csak becsülni lehet. Ezen felül belátható, hogy a vetítés művelete miatt számos, számunkra fontos geometriai jellemző is torzul. Ezen felül a térben egyenlő méretű objektumok a képen eltérő méretűek lesznek, ha a kamerától vett távolságuk más.

A vetítés során nem csak a méretek, hanem a szögek is megváltoznak, ami a geometriai alakzatok, formák torzulásához vezet. Ezen torzulásnak egy rendkívül szemléletes példája az eltűnő pont fogalma. Mint azt tudjuk, a párhuzamos egyenesek a végtelenben metszik egymást. Azonban, ha ezeket a párhuzamos egyeneseket egy kamera segítségével egy képre vetítjük, akkor ez a végtelen távol lévő metszéspont is rajta lesz a képen. Ez azt jelenti, hogy egy végtelen távol lévő pont vetülete lehet véges, mely esetben ezt a vetületet eltűnő pontnak nevezzük.

13.1. ábra. *A 3D geometriai jellemzők torzulása (bal) és az eltűnő pont (jobb).*

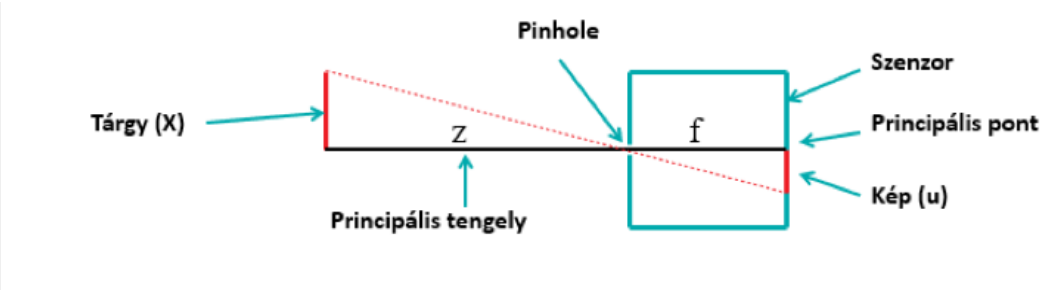
Látható tehát, hogy az objektumok néhány jelentős tulajdonságát lehetetlen egyetlen képből meghatározni (habár az árnyékok felhasználhatók becslések elvégzésére). Ha ezekre az információkra mégis szükségünk van, akkor lehetőség van olyan képalkotó eszközöket használni, amelyek képesek a kép minden pixeléhez mélység információt is számítani (RGB-D szenzorok). Ezek az eszközök azonban általában lényegesen költségesebbek, mint a közönséges kamerák (azonos minőség mellett nagyjából 5-szörös árkülönbség), így ez nem mindig észszerű.

Szerencsére van egy másik lehetőségünk, ugyanis az emberi látáshoz hasonlóan ki tudjuk használni, hogy kettő vagy több kamera/felvétel segítségével visszaállítható az eredeti háromdimenziós tér egy része. A háromdimenziós számítógépes látás területe ennek a feladatnak a minél pontosabb és hatékonyabb megoldásával foglalkozik. A jelenlegi fejezetben ezt a területet fogom tárgyalni.

Fontos megjegyezni, hogy a legalább kettő felvétel azért szükséges, mert ugyanazt a jelenetet egy más pozícióból levetítve újabb információkat kapunk az eredeti jelenet geometriájáról. Ezeket az információkat egymással megfeleltetve vissza tudjuk állítani az egyes vetítések során elveszett és torzult információt. Érdemes belátni, hogy mindehhez nem feltétlenül szükséges két kamera, elég, ha egy kamerával készítünk két felvételt egymás után, különböző pozíciókból. Ebben az esetben azonban rendkívül fontos, hogy maga a jelenet ne változzon meg a két felvétel között, az ugyanis a rekonstrukció eredményét meghamisítja. Ha az objektumok mozgását/változását nem tudjuk korlátozni, akkor mindenképp érdemes két szinkronizált kamerát alkalmazni.

13.2. Pinhole kamera modell

A háromdimenziós látás során tehát a feladatunk a kamera vetítése során elveszett és torzult információk rekonstrukciója. Ahhoz, hogy ezt a feladatot elvégezhessük, először meg kell érteni magát a vetítés folyamatát, vagyis fel kell állítanunk egy matematikai modellt. Ezt követően egy konkrét kamerarendszer esetén méréseket kell végeznünk, hogy a korábban felállított modell ismeretlen paramétereit meghatározhassuk. Ezt a lépést nevezzük kamerakalibrációnak. Az alábbi alfejezetben a kalibráció két fontos esetét vizsgáljuk meg: az első esetben egyetlen kamera vetítésének paramétereit határozzuk meg, míg a második esetben egy kamerarendszeren belüli kamerák relatív pozícióinak meghatározására törekszünk.

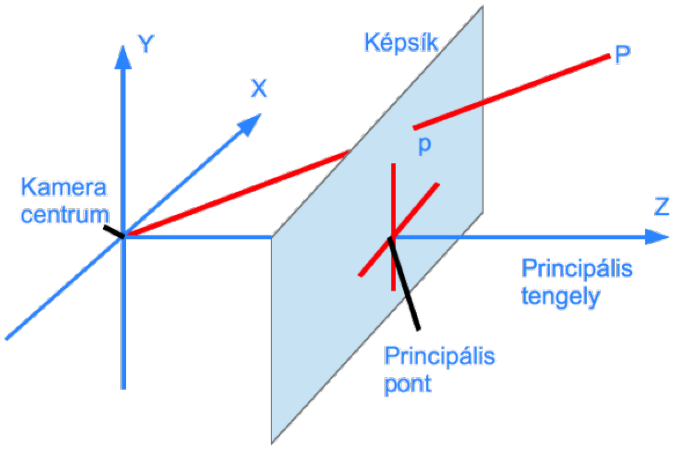


13.2. ábra. *A pinhole kamera fizikai modellje.*

A számítógépes látásban gyakran használt a pinhole kameramodell. A pinhole kamera egyszerűen elképzelhető úgy, mint egy doboz, aminek az egyik oldalán van egy kis lyuk, amin keresztül fény képes beáramlani. A lyukon beérkező fény hatására a doboz ellenkező oldalán egy fordított állású kép keletkezik. Valódi kamerák esetén itt helyezkedik el a fényszenzor. A valódi kamerák további jelentős különbsége, hogy egyetlen kis lyuk helyett egy lencsét alkalmaznak, ami a párhuzamos fénysugarakat egy helyre fókuszálja, így képes a pinhole-t helyettesíteni. A lencse alkalmazásának előnye, hogy lényegesen több fényt ereszt be, mint a pinhole, azonban geometriai torzítást okoz a képen. A pinhole kameramodell az alábbi egyenletek segítségével írható le:

$$u = f_x \frac{x}{z} + p_x; \quad v = f_y \frac{y}{z} + p_y; \tag{13.1}$$

Ahol u és v a pixelek koordinátái, x, y, és z az objektum térbeli koordinátái, f a kamera fókusztávolsága, p pedig a principális pont. Mielőtt azonban a kameramodellt tovább tárgyalnánk, először egy kis kitérőt kell tennünk. A kamerakalibráció során voltaképpen a pinhole kameramodell egyes elemeinek (fókusztávolság, principális pont stb.) numerikus becslését fogjuk elvégezni. A numerikus becslések során azonban rendkívül fontos, hogy a problémát olyan formában fogalmazzuk meg, hogy a becslést majd minél könnyebb legyen elvégezni. A kalibráció – és általánosságban – a geometriai jellegű problémák esetén éppen ezért szokványos az úgynevezett homogén koordináták használata.



13.3. ábra. *A pinhole kamera geometriai modellje. Érdemes megjegyezni, hogy ezen a modellen a képsík már a pinhole előtt található: ily módon álló képet kapunk. Ez természetesen csak matematikai egyszerűsítés.*

13.2.1. Homogén koordináták

A homogén koordináták használata az úgynevezett projektív geometriában elterjedt. A projektív geometria az euklideszi geometria egy kiterjesztése, amely lényegesen több transzformációt enged meg. Míg az euklideszi geometria csak merev transzformációkat (eltolás, elforgatás) enged meg, addig a projektív geometria megengedi az objektumok irányfüggő skálázását, nyírását, valamint a projekció műveletét is. A projektív geometria során az euklideszi síkot/teret egy újabb dimenzióval egészítjük ki, így a projektív sík 3, a tér pedig 4 dimenzióval írható le. A két geometria közötti áttérés a következő egyenletekkel írható le.

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \qquad \begin{pmatrix} X \\ Y \\ W \end{pmatrix} \rightarrow \begin{pmatrix} \frac{X}{W} \\ \frac{Y}{W} \end{pmatrix} \qquad \begin{pmatrix} aX \\ aY \\ aW \end{pmatrix} = \begin{pmatrix} X \\ Y \\ W \end{pmatrix} \tag{13.2}$$

Az első két egyenlet az euklideszből projektív geometriába történő át- és visszatérést írja le. Az át- és visszatérési szabályok egy érdekes következménye, hogy ha egy homogén koordinátákkal leírt pontot egy tetszőleges nem nulla skalárral szorzok, akkor a visszatérés után az ugyanahhoz az euklideszi ponthoz fog tartozni. Ezt a skálainvarianciás tulajdonságot fejezi ki a harmadik egyenlet. Ebből következik, hogy egy euklideszi ponthoz tartozó homogén koordináták egy egyenes mentén helyezkednek el, amely a W=1 síkot pont a pont euklideszi koordinátáiban metszi.

A homogén koordináták használatának két fontos előnye van. Ezek közül a legfontosabb, hogy a pinhole kameramodell összefüggései ebben a koordinátarendszerben lineárisak lesznek. A másik rendkívül fontos tulajdonság a skálainvariancia, amely nagymértékben meg fogja könnyíteni a numerikus becslések elvégzését. A homogén koordináták egy érdekessége, hogy létezik az $(x, y, 0)$ pont, amely az euklideszi síkon a végtelenben van, de a projektív síkon mégis csupa véges koordinátával leírható. Ezt az „irányított végtelen” pontot ideális pontnak nevezzük, és önkalibrációs eljárásoknál fontos szerepe van.

Transzformációk

Mint már említettük, a homogén koordinátákat különböző transzformációk leírására alkalmazzuk. Ehhez azonban érdemes megtárgyalni, hogy pontosan milyen geometriai transzformáció típusok is léteznek. A legalapvetőbb ezek közül az úgynevezett Euklideszi transzformáció, amelyet merev (rigid) transzformációnak is nevezünk, ugyanis csak a forgatás, illetve az eltolás műveleteit engedi

meg. Ennek következménye, hogy egy Euklideszi transzformáció során az objektumok méretei, valamint a szögei is változatlanok. Könnyen belátható, hogy ez a képalkotás során nem így történik, ezért kénytelenek vagyunk ennél megengedőbb geometriai transzformációkat alkalmazni. A következő transzformáció típus a hasonlósági transzformáció, amely a korábban említett két műveleten felül már az uniform (minden irányban egyenlő mértékű) skálázást is megengedi. Ez a transzformáció a méreteket már nem, de a szögeket még mindig megtartja. A két transzformáció típus a homogén koordináták segítségével az alábbi módon fejezhető ki:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \qquad \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} ar_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & ar_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & ar_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{13.3}$$

Ahol r_{ij} a rotációs mátrix egy eleme, t_i pedig az eltolás vektoré, a pedig az uniform skálázás paramétere. Megjegyzendő, hogy a rotációs mátrix oszlopai egy hosszúak és páronként merőlegesek, vagyis a rotációs részmátrix egy úgynevezett ortogonális mátrix.

A következő transzformáció típus az affin transzformáció, amely két újabb műveletet - az irányfüggő skálázást és a nyírást - enged meg. Az affin transzformáció a szögeket már nem, de a párhuzamosságot még megtartja, így ez még mindig kevéssé megengedő, mint a vetítés folyamata, ahol ez sem marad meg. Az utolsó típus a projektív transzformáció, amiben már az úgynevezett perspektív vetítés művelete is szerepel, így ez a típus csak a metszéseket tartja meg, a párhuzamosságot nem. A jelenleg tárgyalt két transzformáció típus képletei a következők:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & t_1 \\ a_{21} & a_{22} & a_{23} & t_2 \\ a_{31} & a_{32} & a_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \qquad \begin{pmatrix} wx' \\ wy' \\ wz' \\ w \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & t_1 \\ a_{21} & a_{22} & a_{23} & t_2 \\ a_{31} & a_{32} & a_{33} & t_3 \\ a_{41} & a_{42} & a_{43} & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{13.4}$$

Ahol a korábbiakkal ellentétben a mátrixok az a_{ij} -vel jelölt tetszőleges elemekből épülnek fel.

13.2.2. Vetítés mátrixa

A homogén koordináták bevezetése után felírhatunk a pinhole kameramodell vetítését egyetlen lineáris mátrixszorzás segítségével:

$$\begin{pmatrix} wu \\ wv \\ w \end{pmatrix} = \begin{pmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = Ax \tag{13.5}$$

Ahol f a fókusztávolság, p a principális pont, u és v a pixelkoordináták, A pedig az úgynevezett kameramátrix. Érdemes észrevenni, hogy a z koordinátával történő osztás majd csak az euklideszi síkra történő visszatéréskor fog megtörténni. Fontos megjegyezni, hogy ez az egyenlet akkor igaz így, ha a pont térbeli koordinátái a kamera koordinátarendszerében lettek megadva. A kamera koordinátarendszerének középpontja tipikusan a pinhole, x és y tengelyei a képsíkkal egybeesnek, z tengelye pedig a principális tengely irányába mutat.

A háromdimenziós térben létezik azonban egy másik koordinátarendszer, amit világ koordinátarendszernek szokás nevezni, és a háromdimenziós pontok koordinátái általában ebben a koordinátarendszerben vannak megadva. A világ koordinátarendszer általánosságban az adott alkalmazástól és szituációtól függ. Előfordulhatnak olyan esetek, amikor valamilyen fizikai objektumhoz rögzített, de gyakran szabadon megválaszthatjuk. Utóbbi esetben gyakran célszerű úgy megválasztani, hogy a kamera koordinátarendszerével megegyezzen, de bőségesen akadnak e szabály alól kivételek.

$$\begin{pmatrix} u \\ v \end{pmatrix} \leftarrow \begin{pmatrix} wu \\ wv \\ w \end{pmatrix} \leftarrow A \begin{pmatrix} y \\ y \\ z \end{pmatrix} \leftarrow \begin{pmatrix} R & t \end{pmatrix} \begin{pmatrix} X \\ Y \\ W \end{pmatrix} \tag{13.6}$$

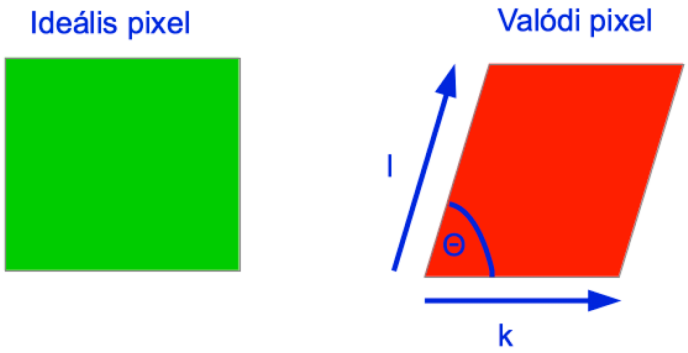
A gyakorlatban a világ koordinátarendszer nem feltétlenül egyezik meg a kamera koordinátarendszerével, így a kettő közötti transzformációt is meg kell határozni a kalibráció során. Szerencsére két koordinátarendszer közti áttérés egy egyszerű euklideszi transzformáció, így csak egy R elforgatás és egy t eltolás együtteséből áll. Érdemes észrevenni, hogy az így kapott ismeretlen paramétereink két külön csoportra oszthatók. Az egyik csoportba az A kameramátrix elemei tartoznak: ezek a paraméterek kizárólag az adott kamera belső felépítésétől függenek, így ezeket belső (angolul: intrinsic) paramétereknek nevezzük.

A második csoport az R és t elemeit foglalja magába, vagyis egyáltalán nem függ a kamera belső tulajdonságaitól, hanem kizárólag a konkrét elrendezés tulajdonságaitól függ. Éppen ezért ezeket külső (angolul: extrisic) paramétereknek nevezzük. A két paramétercsoport külön-külön meghatároz egy-egy geometriai transzformációt, kettőjük kompozíciója pedig a teljes vetítés mátrixát (P):

$$P = A[R \;\; t] \tag{13.7}$$

13.2.3. Valódi optikák

A valóságban a kamerákban egy valamelyest módosított elrendezést használunk, ugyanis a pinhole túlságosan kicsi ahhoz, hogy azon a jó minőségű képalkotáshoz elegendő fény áramoljon be. Ha a pinhole méretét növelnénk, akkor pedig a fénysugarak nagyobb beesési szöge miatt homályos képet kapnánk. Éppen ezért a kamera bemeneti nyílásához egy lencsét teszünk, amely egy pontba fókuszálja a párhuzamosan beérkező fénysugarakat, így helyettesíti a pinhole-t, mérete azonban már elég nagy ahhoz, hogy megfelelő mennyiségű fényt engedjen át.

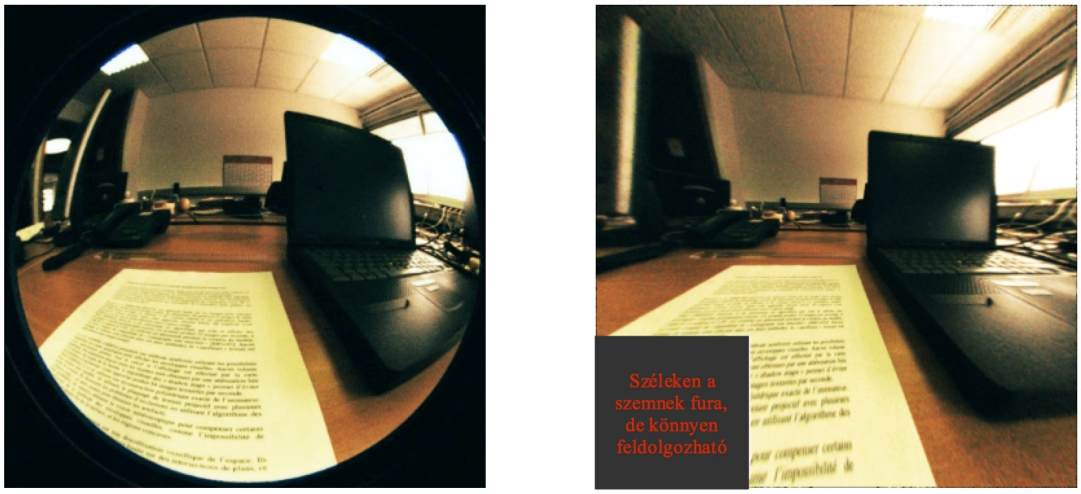


13.4. ábra. *A valódi pixelek egyes kamerákban nem feltétlenül négyzetes alakúak. Lehet eltérés a pixel oldalarányaiban (főleg régi videókamerák esetében), valamint a két oldal szöge is különbözhet a 90 foktól.*

A lencse hozzáadása azonban komplikációt okoz: a kamera látóterének a szélső részeiből érkező képsugarak már nem párhuzamosan fognak beérkezni a lencsére, így másképp fognak megtörni, mint a párhuzamosan érkező sugarak. Ennek következményeképp ezek a fénysugarak a fényérzékelő szenzortömbre más pozícióban fognak becsapódni, így a képen is arrébb kerülnek. Ezt a jelenséget radiális torzításnak hívjuk, és gyakori jelenség, különösképpen széles látószöggel rendelkező kamerák esetén.

13.3. Kalibráció

A háromdimenziós rekonstrukció elvégzéséhez első lépésként meg kell határozni a kamera vetítésének paramétereit. Ehhez bevezetésképpen először részletesen meg kellett értenünk a vetítés



13.5. ábra. *A radiális torzítás (bal) és a torzításmentesített kép (jobb).*

matematikáját, melynek során megértettük, hogy a vetítés tulajdonképpen két részre bontható. Az egyik rész a kamera világban lévő helyzetétől, a másik pedig a kamera belső tulajdonságaitól függ. A jelenlegi alfejezetben ismertetett módszerek első sorban (de nem kizárólag) ez utóbbi belső paraméterek meghatározására szolgál.

Felmerülhet azonban a kérdés, hogy hogyan is határozhatjuk meg ezeket a paramétereket. A válasz egészen egyszerű: mivel ismerjük a térbeli pontok és a képen lévő vetületük közötti matematikai összefüggést, ezért nincs más dolgunk, mint számtalan mérés elvégzése után numerikusan a vetítés ismeretlen paramétereit megbecsülni. Képek esetén ez a gyakorlatban azt jelenti, hogy készítünk egy kalibrációs objektumot, amin előre ismert pozíciókba könnyen felismerhető markereket helyezünk el. Ezt követően a kalibrációs objektumról képeket készítünk, és a képen meghatározzuk az egyes objektumok pozícióját, és az így kapott pontpárokból végezzük el a becslést.

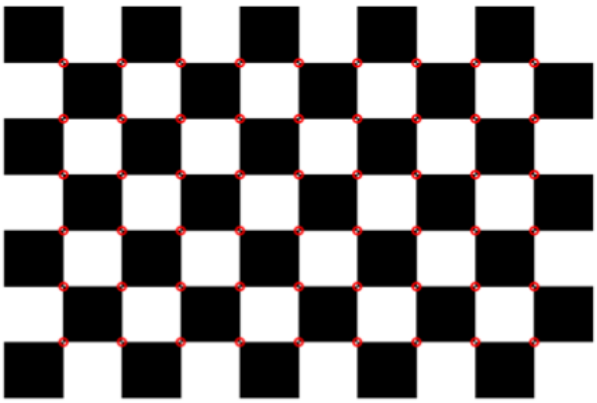
A használt kalibrációs objektumoktól függően a kalibráció több formáját különböztetjük meg. Matematikai szempontból a legegyszerűbb eset, ha a kalibrációs objektumon található markerek teljesen általános elrendezésűek, vagyis nem esnek egy síkba vagy egyenesre. Ebben az esetben háromdimenziós kalibrációs objektumról beszélünk. Létezik azonban két- és egydimenziós kalibrációs objektum is: itt azonban ahogy a markerek elrendezése egyre speciálisabb, a kalibráció mögött rejlő matematika egyre bonyolultabbá válik.

A kalibráció egy speciális esete az önkalibráció, amikor egyáltalán nincs kalibrációs objektum, hanem a kalibrációt képek közötti megfeleltetések segítségével végzik el. Ekkor a képeken előforduló könnyen felismerhető és azonosítható, úgynevezett természetes markereket használjuk fel. Az önkalibráció egyik fontos limitációja, hogy kizárólag a belső paraméterek határozhatók meg a segítségével. További hátránya, hogy legalább három különböző kép készítése szükséges, míg a kalibrációs objektumokat használó módszerek esetén elvileg egy is elegendő.

A jelen fejezetben a három- és kétdimenziós kalibrációs objektumokat használó kalibrációs módszereket fogjuk részletesebben megismerni. Mindkét esetben a kalibráció lépései megegyeznek:

1. Számos különböző kép készítése a kalibrációs objektumról
2. Markerek megkeresése és azonosítása az egyes képeken
3. A teljes vetítés mátrixának meghatározása
4. A belső és külső paraméterek szétválasztása
5. A becslés finomítása

A legtöbb esetben a kalibrációs objektumon valamilyen ismétlődő formák (pl. négyzetek) találhatók, ahol az egyes markerek az ismétlődő formák által meghatározott sarokpontok. Kétdimenziós kalibráció esetén szinte minden esetben egy sakktáblaszerű kalibrációs objektumot használnak, ami miatt ezt az esetet gyakran nevezik „sakktáblás" kalibrációnak. Ennek a megoldásnak nagy előnye, hogy rendkívül egyszerű és olcsó a kalibrációs objektumot nagy pontossággal előállítani a fejlett nyomtatóknak köszönhetően. Háromdimenziós esetben is gyakran használnak egymással merőlegesen elhelyezett sakktábla-szerű objektumokat, itt azonban már nehezebb a pontosságot a két objektum illesztésénél megoldani. Az ismétlődő elrendezés másik nagy előnye, hogy ha a világ koordinátarendszert a kalibrációs objektum egy kitüntetett pontjába (pl. sakktábla bal felső sarka) választjuk, akkor az egyes markerek térbeli koordinátáit könnyedén előállíthatjuk bonyolult mérések nélkül.



13.6. ábra. *Egy sakktáblaszerű kalibrációs objektum és annak markerei.*

Ezt követően számos képet készítünk a kalibrációs objektumokról különböző pozíciókból. Mivel mérések segítségével végzünk numerikus becslést ezért minél több mérés áll rendelkezésre annál pontosabb lesz a becslés. Fontos megjegyezni, hogy ez csak a belső paraméterek meghatározására igaz. Ugyanis a képeket különböző pozíciókból készítjük, tehát minden kép esetén a kamera és a kalibrációs objektum relatív pozíciója – vagyis a külső paraméterek - másak, így nem tudunk több képet felhasználni ezek pontosítására. A kamera maga viszont nem változik, így a belső paraméterek minden kép esetén megegyeznek. Ha a külső paraméterek becslését szeretnénk pontosítani, akkor célszerű a kalibrációs objektumon található markerek számát növelni.

A markerek leggyakrabban valamilyen sarokszerű pontok a kalibrációs objektumon, ezért célszerű valamilyen ismert sarokdetektáló módszert (pl. KLT, Harris) alkalmazni. Ezen módszereknek azonban alapvető hátránya, hogy a megtalált sarokpozíciók nem szubpixeles pontosságúak, ami a kalibrációs probléma rossz kondíciószáma miatt nagy hibákat okozhat a becslésben. Éppen ezért sakktáblaszerű elrendezések esetén gyakori megoldás, hogy sarkok helyett éleket detektálunk, amelyekből a Hough transzformáció módszere segítségével egyeneseket állapítunk meg. Mivel a Hough transzformáció során az egyeneseket paraméteresen kapjuk meg, ezért ebből könnyedén kiszámolhatjuk az egyenesek metszéspontjait. Az így kapott sarokpontok pontossága már pixel alatti szinten mérhető.

13.3.1. A vetítés meghatározása

Ezt követően az $\vec{u} = P\vec{X}$ lineáris egyenletrendszerben már az $\vec{u}$ és az $\vec{X}$ értékei ismertek, így csak a P meghatározása van hátra. Itt viszont az a probléma adódik, hogy a lineáris egyenletrendszereket akkor tudjuk megoldani, ha az $\vec{X}$ vektor helyén álló változó az ismeretlen, a mi esetünkben viszont a P mátrix az. Ez a probléma azonban egy egyszerű egyenletrendezés segítségével orvosolható és az adott képen lévő összes pontpárra felírva kapott egyenletrendszer a legkisebb négyzetek módszerével megoldható. Az egyenlet átrendezéséhez először kifejezzük az u koordináta értékét az alábbi módon:

$$\begin{pmatrix} wu \\ wv \\ w \end{pmatrix} = \begin{pmatrix} p_1^T \\ p_2^T \\ p_3^T \end{pmatrix} \vec{X} \quad \rightarrow \quad u = \frac{wu}{w} = \frac{p_1^T \vec{X}}{p_3^T \vec{X}} \tag{13.8}$$

Ahol p_1^T a P projekciós mátrix első sora. Ha hasonló módon a v koordinátát is kifejezzük, akkor az alábbi egyenletrendszert kapjuk:

$$\begin{aligned} u p_3^T \vec{X} &= p_1^T \vec{X} \\ v p_3^T \vec{X} &= p_2^T \vec{X} \end{aligned} \tag{13.9}$$

Mely mátrixos formába rendezve:

$$\begin{pmatrix} \vec{X} & \vec{0} & -u\vec{X} \\ \vec{0} & \vec{X} & -v\vec{X} \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} = G\vec{p} = \vec{0} \tag{13.10}$$

Az egyenletrendszert a többi pontpárra felírva a G mátrix újabb sorokkal bővül. Érdemes megjegyezni, hogy a legkisebb négyzetes megoldás során kénytelenek vagyunk önkényesen megválasztani a P vetítés mátrix normáját, különben nem tudnánk egyértelműen megoldani az egyenletrendszert. Emlékezzünk azonban vissza, hogy a P mátrix homogén koordinátákon működik, amik a skalárral való szorzásra invariánsak. Ennek következtében bárhogy is választjuk meg a P mátrix normáját, mivel az minden skálafaktor mellett ugyanazt a geometriai transzformációt fogja végrehajtani.

Ezt követően a megkapott P vetítés mátrixot egy A kameramátrixra és egy $[Rt]$ merev transzformációra kell felbontani. A háromdimenziós kalibráció esetében ez a forgatásmátrix és a kameramátrix speciális tulajdonságainak köszönhetően a jól ismert QR felbontás segítségével könnyedén elvégezhető.

Érdemes megjegyezni, hogy a sakktáblás esetben a P nem határozható meg, mivel a markerek mind egyetlen síkba esnek, így a probléma rosszul kondícionálttá válik. Ez könnyen kezelhető, ha a vetítés mátrixa helyett az ún. H homográfia mátrixot határozzuk meg. Ehhez feltételezzük, hogy a sakktábla a $z = 0$ síkban van, és így felírva a vetítés egyenletét megkaphatjuk ezt a mennyiséget:

$$\vec{u} = \begin{pmatrix} r_1 & r_2 & r_3 & t \end{pmatrix} \begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} r_1 & r_2 & t \end{pmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} = H\vec{X} \tag{13.11}$$

Innen a H ugyanúgy a legkisebb négyzetek módszerével becsülhető. Ezt követően a belső és külső paraméterek szintén előállíthatók, ehhez azonban a QR felbontásnál lényegesen bonyolultabb matematikát kell alkalmazni.

Fontos megjegyezni, hogy ez az egyenlet ugyanúgy megoldható a legkisebb négyzetek módszerével, azonban mivel a kimeneti vektor csupa nulla elemből áll, ezért a Total Least Squares (TSL) módszert kell alkalmazni.

13.3.2. A geometriai hiba

Fontos megjegyezni, hogy a képenként eltérő külső paraméterek miatt több kalibrációs kép használata esetén ezeket a becsléseket képenként külön-külön kell elvégeznünk. A külön becslésekből viszont - a hibákat leszámítva – megegyező kameramátrixokat kapunk, melyek átlagát véve egy meglehetősen jó becslést kapunk a belső paraméterekre. Fontos azonban megjegyezni, hogy ezek a becslések egy algebrai egyenletrendszer megoldásából származnak, úgy, hogy az egyenlet hibáját

igyekeztünk minimalizálni. Ez az úgynevezett algebrai hiba azonban nehezen értelmezhető, nehéz megmondani, hogy pontosan milyen szempontból optimális a kapott megoldás.

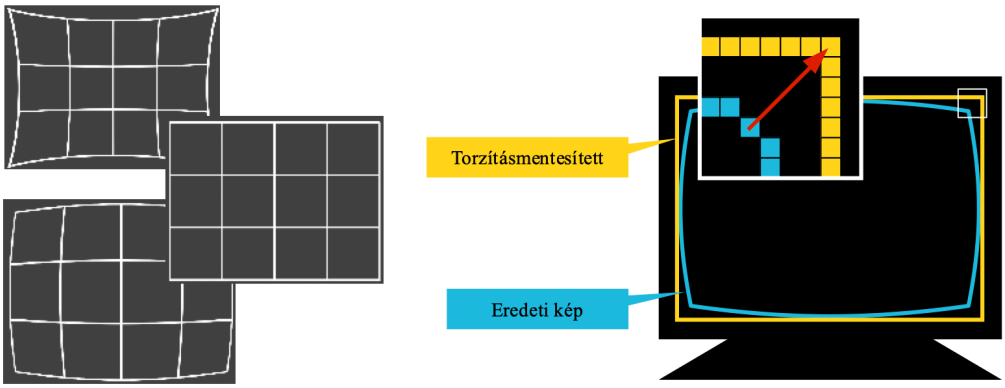
Éppen ezért bevezetjük az úgynevezett geometriai hibát, ami azt adja meg, hogy az ismert térbeli pontokból a becsült projekció segítségével számolt vetületek milyen távolságra vannak a képen megtalált markerektől. Ez a hiba meglehetősen szemléletes és könnyen megérthető, azonban a felírása erősen nemlineáris:

$$E_G = \sum_i ||\vec{u}_i - f(X_i, P)||^2 \tag{13.12}$$

Ahol u_i és X_i az i-edik 2D-3D pontpár, f pedig a vetítés pontos függvénye. Szerencsére azonban az algebrai és a geometriai hibák minimumhelyei meglehetősen közel helyezkednek el egymáshoz, így az algebrai hibát minimalizáló kameramátrix jó kezdeti értékként szolgálhat egy iteratív, gradiens alapú optimalizáló módszernek. Az ilyen módszerek rendkívül hatékonyak bonyolult függvények esetén is, amennyiben a valódi optimum közeléből tudnak indulni. A kamerakalibráció finomító lépése során gyakran használjuk a gradiens, a Newton és a Levenberg-Marquardt módszereket.

13.3.3. Lencsetorzítás meghatározása

Fontos még megjegyezni a geometriai torzítások fontos szerepét. Ezek a torzítások a lencse tulajdonságaiból vagy hibáiból származnak, így a kamera belső paraméterei közé tartoznak. A hatásuk azonban nemlineáris, így a kameramátrixba nem vehetők be, valamint nem is becsülhetők könnyedén. A kalibráció utolsó, finomító lépése során azonban már egy bonyolult, nemlineáris függvény minimumát keressük, amihez a lencsetorzítások hatása egyszerű módon hozzáadható. Így a finomító lépés felhasználható a torzítások paramétereinek meghatározásához is.



13.7. ábra. *A lencsetorzítást gyakran egy lookup table segítségével korrigáljuk. Meghatározás után minden pixel esetére meghatározzuk azt, hogy a torzításmentesítés után hova kellene kerülnie, majd e térkép segítségével már gyorsan elvégezhetjük a képek torzításmentesítését.*

13.4. Sztereó kalibráció

Amint azt a korábbi előadásban megállapítottuk, a háromdimenziós rekonstrukcióhoz több kamerára, vagy legalábbis több különböző pozícióból elkészített felvételre van szükség. Ezen felül szükség van arra is, hogy a különböző kamerák vagy felvételek közti geometriai kapcsolatot ismerjük. Vannak olyan esetek, ahol valamilyen külön szenzor segítségével lehetőségünk van arra, hogy a felvételek közti kapcsolatot más forrásból megtudjuk (főleg a mobil robotikában gyakori ez az eset), azonban gyakran csupán a kamerák által készített képekre hagyatkozhatunk. A jelenlegi alfejezetben azt vizsgáljuk meg, hogy hogyan lehet kalibrációs eljárások segítségével kamerák vagy felvételek relatív helyzetét meghatározni.

A lehetséges módszerek feltárásához azonban érdemes felismerni, hogy a feladat természete és következésképp a nehézsége nagymértékben függ az elrendezéstől. A korábbi fejezetekben is megkülönböztettük azt a két esetet, amikor két kamera készít külön pozícióból két felvételt, és amikor egy kamera mozdul el a két felvétel között. Az előbbi szituációban egy fix kamerarendszerről beszélhetünk, ahol az egyes kamerák nem mozdulnak el egymáshoz képest a felvételek készítése során, ezért elég egyszer, a kezdetekkor kalibrálni. A második szituációban azonban a kamera folyamatosan mozog, így minden két felhasználandó felvétel között el kell végezni a mozgás becslését.

Fix kamerarendszerek esetén rendkívül szerencsés esetünk van, ugyanis a kamerák belső kalibrációját (amit egyébként is el kell végezni) egyszerre el tudjuk végezni az összes kamerával. Ilyenkor csak arra kell ügyelni, hogy a kalibrációs objektum minden kamerán látszódjon. Emlékezzünk vissza, hogy az összes kalibrációs objektumot használó eljárás esetén nem csak a kamerák belső paraméterei, hanem a külső paraméterek (vagyis a kalibrációs objektum és a kamera közti transzformáció) is előálltak minden kalibrációs képhez.

Ha viszont egy adott képen ismerjük az egyes kamerák és egy kitüntetett pont közti transzformációt, akkor ebből könnyedén számolható az egyes kamerák közti transzformáció. Sőt, mivel az egyes képek között a kamerák relatív helyzete nem változik, ezért az összes kép felhasználható ennek becslésére. Ebből az következik, hogy fix kamerarendszerek esetén nincs szükség a kamerarendszert külön kalibrálni, hiszen az egyébként is szükséges belső kalibráció során a relatív pozíciók könnyedén előállíthatók.

13.4.1. Epipoláris geometria

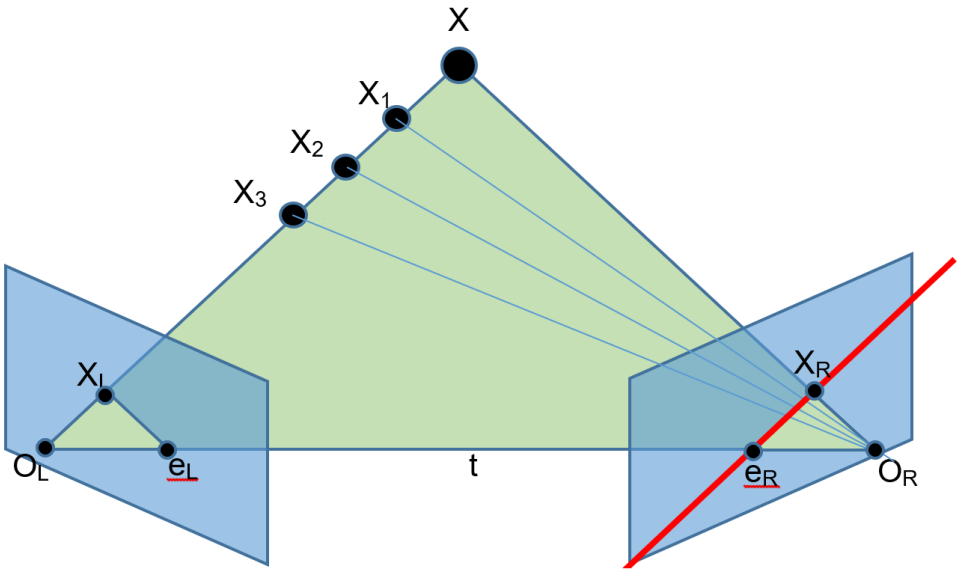
A helyzet azonban teljesen más mozgó kamerák esetén. Ekkor a kamera kalibrációját egyszer a használat elején elvégezzük, majd a kamerát a számunkra érdekes térben mozgatva kívánunk egy rekonstrukciót előállítani. Ekkor már nincs a térben olyan általunk gyártott kalibrációs objektum, amit az egyszerű kalibrációhoz felhasználhatnánk. Ez azt jelenti, hogy az elmozdulás becslését a képen előforduló természetes markerek segítségével kell elvégeznünk. Ez problémás feladat, ugyanis a természetes markerek térbeli koordinátáit nem ismerjük (ha ismernénk, akkor kész lenne a rekonstrukció), így csak arra hagyatkozhatunk, hogy ha ugyanazt a markert mindkét képen megtaláljuk, akkor ezekre a pontpárokra képesek leszünk egy egyenletet felírni.

Ehhez azonban először meg kell vizsgálnunk magát a geometriai elrendezést. Az egyszerűség kedvéért szorítkozzunk az úgynevezett sztereó elrendezésre, ahol a mozgó kamera két pozícióban ábrázolt: ezeket hívjuk jobb és bal kameráknak. A bal kamera középpontja O_L, és egy térbeli X pont a képsíkját X_L pontban metszi. A jobb kamera középpontja O_R, és az X pont a képsíkját az X_R pontban metszi. A két kameraközéppont közti eltolást a t vektor, a képsíkok közti elforgatást pedig az R mátrix írja le.

Érdemes észrevenni, hogy ha ismerjük az O_L és X_L pontokat, de X-et nem (ahogy ez a valóságban így is van), akkor meg tudjuk határozni azt az irányt, amerre X a bal kamerától található, de a távolságát nem. Ez azt jelenti, hogy egy térbeli egyenes mentén végtelen sok jelöltünk van az X pontra. A projekció azonban egy térbeli egyenest a képsíkon szintén egyenesbe képez le, ezért X_L összes lehetséges párja a jobb kamera képén is egy egyenes mentén helyezkedik el. Ezeket az egyeneseket hívjuk epipoláris egyenesnek. Fontos tudni, hogy az összes epipoláris egyenes egy pontban metszi egymást, ezeket hívjuk epipoláris pontnak (e_L és e_R). Továbbá az epipoláris pont az a pont, ahol a két kameraközéppontot összekötő szakasz metszi az egyes képsíkokat.

Fontos észrevenni az ezekből következő tulajdonságot: a vektor, amely a két kamerát összeköti (t), a vektor, ami a bal kamera középpontjából az X_L pontba mutat, és ami a jobb kamera középpontjából az X_R pontba mutat ugyanabba a térbeli síkba esik. Ez persze csak akkor igaz, ha X_L és X_R ugyanannak az X pontnak a képe. Ha három vektor egy síkba esik, az azt jelenti, hogy ha bármelyik kettőt kiválasztom és kiszámolom azok vektoriális szorzatát, akkor az a harmadik vektorra merőleges lesz. Ebből az észrevételből azonban felírható az alábbi egyenlet:

$$x_L^T(t \times R^T x_R) = x_L^T([T\times]R^T)x_R = x_L^T E x_R = 0 \tag{13.13}$$



13.8. ábra. *Az epipoláris elrendezés.*

Ahol $[T\times]$ a t vektorral történő vektoriális szorzás mátrixa, E pedig az úgynevezett esszenciális mátrix. Érdemes megjegyezni, hogy az X_R vektort a forgatás mátrix segítségével a bal kamera koordinátarendszerébe kellett forgatni, ugyanis minden vektort ugyanabban a koordinátarendszerben kell felírni ahhoz, hogy az összefüggés értelmes legyen. Kaptunk tehát egy egyenletet, ahol szükségünk van egy pontpárra a két kamera közt és ismeretlenek a forgatás és eltolás paraméterek. Egy apróbb probléma, hogy X_L és X_R nem a pixelben mért koordináták, hanem a kamera koordinátarendszerében értelmezett vektorok. A kettő között az áttérést a kameramátrix adja meg:

$$x_L = A_L^{-1} u_L \tag{13.14}$$

Ezt behelyettesítve:

$$u_L^T A_L^{-T} E A_R^{-1} u_R = u_L^T F u_R^T = 0 \tag{13.15}$$

Ahol F a fundamentális mátrix. Amennyiben a kameramátrixokat a belső kalibráció során meghatároztuk, akkor az F kiszámolása után E könnyedén meghatározható. Azonban ezt követően az E mátrixból a forgatás és az eltolás értékeit még ki kell nyerni, ami egyáltalán nem triviális. A felbontás során a forgatás mátrixra két külön lehetőségünk adódik, míg az eltolásra egy, de ismeretlen előjellel. Ezekből négy lehetséges kombináció adódik, amikből szerencsére egyszerű geometriai megkötésekkel kiválasztható egy lehetséges megoldás.

Egy jelentős probléma azonban megmarad, mégpedig az, hogy a becslés során semmilyen módon nem tudjuk meghatározni az eltolás vektor nagyságát. Ez azt jelenti, hogy tudjuk, hogy merre mozdult el a kamera, és hogy hogyan fordult el, de nem tudjuk megmondani, hogy pontosan mennyit mozgott az adott irányba. Ettől a háromdimenziós rekonstrukciót még el fogjuk tudni végezni, azonban nem lesz információnk a kapott rekonstrukció skálájáról.

13.4.2. Kalibrációs eljárások

A következő fontos lépés az F meghatározása mérésekből. Ennek alapvető módszere, hogy mindkét képen természetes markereket keresünk, majd ezeket valamilyen jellemző leíró (pl. SIFT) segítségével párosítjuk. Megfelelő számú pár esetén fel tudunk írni egy lineáris egyenletrendszert, amiből

F paraméterei számolhatók a legkisebb négyzetek módszerével. A fundamentális mátrix kiszámolására két elterjedt módszer létezik, melyeket 8, illetve 7 pontos módszereknek nevezünk. Ezek – nevükhöz híven – 8, illetve 7 pontpár segítségével számolják ki F értékét.

A 7/8 pontos módszer esetében ugyanabba a problémába ütközünk, mint a belső kalibráció esetében: az egyenletrendszerünk nem olyan alakban állt elő, ahogy azt egyszerűen meg tudjuk oldani. Az ismeretlen paraméterek az F mátrixban vannak, nem pedig a szorzat jobb szélén álló vektorban. Szerencsére a korábbi esethez hasonlóan ez a probléma is könnyedén orvosolható az alábbi átrendezéssel:

$$Af = \begin{pmatrix} u_L^1 u_R^1 & u_L^1 v_R^1 & u_L^1 & v_L^1 u_R^1 & v_L^1 v_R^1 & v_L^1 & u_R^1 & v_R^1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_L^N u_R^N & u_L^N v_R^N & u_L^N & v_L^N u_R^N & v_L^N v_R^N & v_L^N & u_R^N & v_R^N & 1 \end{pmatrix} \begin{pmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{pmatrix} = 0 \tag{13.16}$$

Ahol az f vektor a fundamentális mátrix elemeit tartalmazza. A 8 pontos módszer esetén A mátrixnak 8 sora van: minden pontpárhoz egy. Ebben az esetben azonban egy problémába ütközünk. Ahhoz, hogy az eredeti egyenletet kielégítse, az F mátrixnak szinguláris nak kell lennie, azonban a jelenleg ismertetett becslési módszerrel nem lesz az. Erre a problémára megoldás, hogy megkeressük becslésből adódó F mátrixhoz legközelebbi szinguláris mátrixot. Ehhez mindössze annyit kell tennünk, hogy F szinguláris értékei közül a legkisebbet nullára állítjuk, és az így kapott új mátrix lesz a végső megoldás.

A 7 pontos algoritmus annyiban különbözik, hogy már a kezdeti becsléskor is kihasználja a szinguláris mátrix megkötést, így elég 7 pontpárt felhasználnia a kezdeti becslés előállításához. Ebből az következik, hogy a kezdeti becslés után egy kétdimenziós megoldáshalmazt kapunk egy megoldás helyett. Itt azonban fel lehet használni a szinguláris megkötést, hogy ebből a halmazból a helyes megoldást kiválaszthassuk. Fontos még megjegyezni, hogy a fent ismertetett kalibrációs eljárások meglehetősen érzékenyek a zajra és a skálázásra. Ebből kifolyólag mindig érdemes a felhasznált pontok koordinátáit normalizálva megadni, vagyis a pont koordináták átlagát levonni, majd ezután a szórásukkal leosztani.

További Olvasnivaló

[1] R. Szeliski, *Computer Vision*. Springer London, 2011. DOI: 10.1007/978-1-84882-935-0. cím: https://doi.org/10.1007%2F978-1-84882-935-0.

14. fejezet

3D Rekonstrukció

14.1. Bevezetés

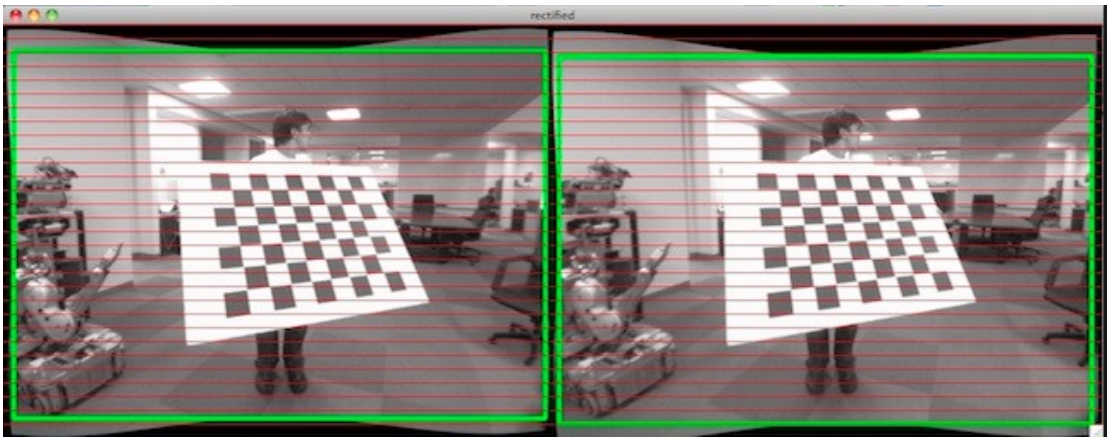
Az előző előadáson megismerkedtünk a képalkotás geometriájának alapvető összefüggéseivel, valamint a kalibrációs eljárások legfontosabb részleteivel. Mindezek a háromdimenziós rekonstrukció alapvető fontosságú részletei, ezen felül bármilyen rekonstrukciós rendszer felépítésének első, megalapozó lépései. A jelenlegi előadás témája azonban már a konkrét rekonstrukció kivitelezése és az ehhez szükséges számítógépes látás algoritmusok ismertetéséről szól.

A háromdimenziós rekonstrukció elvégzését alapvetően négy lépésre oszthatjuk, melyek közül egyet már a korábbi előadáson tárgyaltunk. Ez az első lépés az egyes kamerák és a kamerarendszer kalibrációjának elvégzése. Ezt követően a kamerarendszer kalibrációja során előállt rektifikációs transzformáció segítségével a rekonstrukcióhoz használandó felvételeket transzformáljuk. Ezt követi az egyes képek közti megfeleltetések keresése, majd a kalibráció eredményei és a kapott pontpárok alapján számoljuk a térbeli pontokat. A háromdimenziós tér visszaállítását természetesen követheti számos további feldolgozó lépés, hiszen a rekonstrukció célja a térbeli feldolgozás. Az erre szolgáló algoritmusok működése azonban a következő alfejezet témája lesz.

14.2. Rektifikáció

Ha a kamerák közötti transzformációt meghatároztuk, akkor ez egy újabb lehetőséget teremt a számunkra. Emlékezzünk vissza a sztereó elrendezés során bevezetett epipoláris egyenesekre. Ezek azt az egyenest adják meg, amely mentén egy pont párja található a másik kamera képén. Ezek az egyenesek természetesen minden ponthoz különbözők. Létezik azonban egy olyan kitüntetett kameraelrendezés, ahol minden epipoláris egyenes vízszintes. Ezt az elrendezést sztenderd sztereó elrendezésnek nevezzük, és akkor áll elő, ha a kamerák közti eltolás tisztán vízszintes, és a képsíkjaik közt nincs elfordulás.

Ennek az elrendezésnek az a hatalmas előnye, hogy az egyes képpontok párját nem a teljes másik képen, hanem csak egyetlen sorban kell keresni. Ez általában 2-3 nagyságrendes gyorsulást eredményez. Természetesen a gyakorlatban csak ipari eszközökkel lehetséges egy ilyen elrendezés előállítása, a kalibráció elvégzése azonban lehetőséget ad a sztenderd elrendezés mesterséges előállítására, vagyis a rektifikációra. A rektifikáció során a két képet egy lineáris transzformáció segítségével olyan módon torzítjuk el, hogy az epipoláris egyenesek vízszintesek legyenek. Fontos megjegyezni, hogy lencsetorzítás esetén maguk az epipoláris egyenesek is elgörbülnek, így a rektifikáció során ezeket is ki kell küszöbölni. Ennek következtében a két műveletet gyakran egybevonják, és számos irodalom ezt a közös műveletet is rektifikációnak nevezi.



14.1. ábra. *A rektifikáció művelete.*

14.3. Diszparitás

A készített képek közötti megfeleltetések keresése a rekonstrukció legalapvetőbb művelete. Az az információ, hogy különböző pozíciókból készített képeken különböző helyeken megtalálható ugyanannak a térbeli pontnak a képe pont az az új információ, amire a vetítés során elveszett információ visszaállításához szükségünk van. Éppen ezért a képek közötti párosítás pontossága alapjaiban meghatározza a végső rekonstrukció minőségét. A párosítással kapcsolatban két követelményünk van: egyrészt a párok megtalálása legyen minél pontosabb, másrészt a párosítás legyen a lehető legközelebb a teljeshez, vagyis a lehető legtöbb képpontnak találjuk meg a párját. Ez utóbbi persze a kép véges kiterjedése és a kitakarások miatt nem lehetséges teljes mértékben.

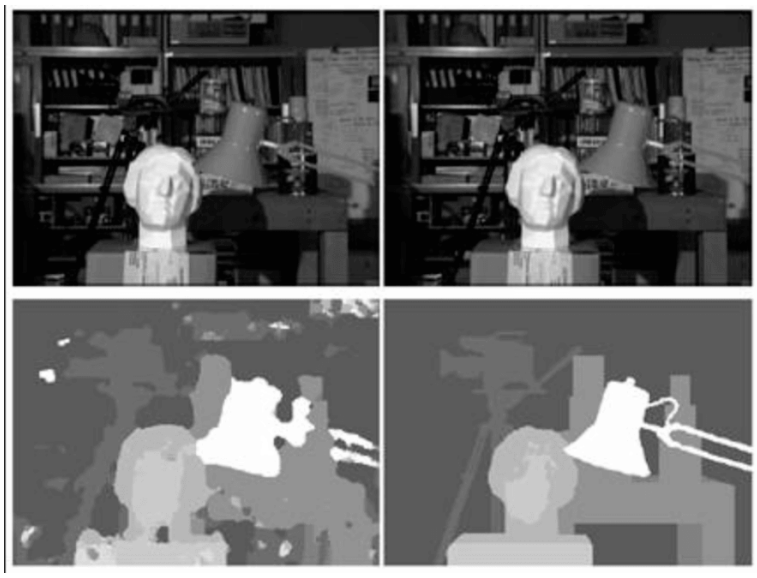
Ahogyan azt már a korábbi fejezetben tárgyaltuk, a rektifikáció elvégzése nagyban megkönnyíti a megfeleltetések keresését. Rektifikált képpár esetén ugyanis garantálni tudjuk, hogy egy adott képpont párja a másik képen ugyanabban a képsorban lesz. Ebből kifolyólag az egyik képen minden egyes pixelhez hozzárendelhetünk egy úgynevezett diszparitás értéket, amelyik azt fejezi ki, hogy a párja hány pixel pozícióval van eltolva hozzá képest. Más kifejezéssel élve a diszparitás a pixel és a párja közti vízszintes irányú távolság. Érdemes megjegyezni, hogy ritkábban, de előfordulnak olyan sztenderd elrendezések, ahol a kamerák egymás alatt/fölött helyezkednek el, ebben az esetben a diszparitás függőleges irányú.

Amennyiben a kép minden pixeléhez meghatározzuk a diszparitás értéket, akkor egy szürkeárnyalatos képet kapunk, amelyet diszparitás képnek hívunk. Ez a kép könnyen értelmezhető az emberi szem számára is, hiszen a diszparitás értéke alapvetően a kamerától vett távolságtól függ. Feltehetően életében egyszer minden ember elkísérletezett már azzal, hogy az egyik szemét becsukva a kinyújtott kezét figyelte, majd a másik szemére váltva észrevette, hogy a kezét kicsit máshol látja. A kísérletet kicsit tovább folytatva észrevehetjük, hogy minél közelebb van a kezünk, annál nagyobb a két szem által észlelt pozíciók között. Innen könnyű belátni, hogy minél nagyobb a diszparitás érték, annál közelebb van az adott pixelhez tartozó tárgy a kamerához.

A diszparitás meghatározására számos módszer létezik, melyeket két nagy csoportba lehet sorolni: sűrű és ritka diszparitás módszerek. A sűrű módszerek a kép szinte minden pixeléhez meghatározzák a diszparitás értékét, míg a ritka módszerek csupán néhány kitüntetett pozícióban számolnak. Bár célunk, hogy a diszparitást minél több pontban meghatározhassuk, a ritka diszparitás módszerek gyakran lényegesen pontosabb eredményeket szolgáltatnak. A ritka módszerek által számított diszparitást lehetséges sűríteni különböző interpolációs technikák segítségével.

14.3.1. Block Matching

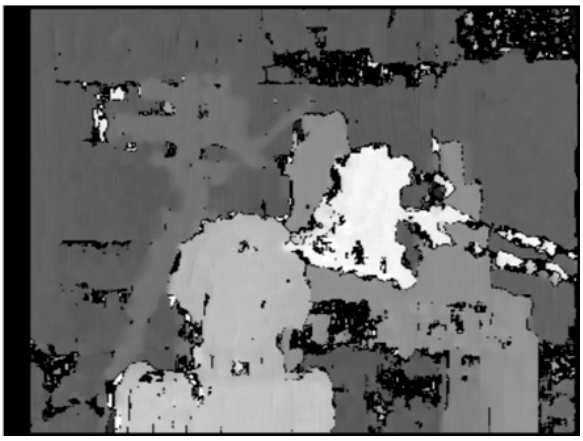
A sűrű diszparitás módszerek egyik legfontosabb családja a block matching (BM) algoritmusra épülő eljárások. A block matching a számítógépes látás egyik alapvető algoritmusa, amelynek



14.2. ábra. *Egy sztereó képpár (fent) és a hozzá tartozó diszparitás képek (alul). A bal oldali kép egy algoritmus eredménye, míg a jobb oldali a valódi diszparitás (ún. ground truth)*

számos alkalmazása létezik, melyekben gyakran valamelyest eltérő néven említik. Az algoritmus lényege, hogy a vizsgált pixel egy környezetét véve végig halad a másik kép megfelelő sorának minden lehetséges pozícióján, és minden lépésnél összehasonlítja az eredeti pixel környezetét a jelenlegi pozícióban található környezettel. A hasonlóság mércéjére számos lehetséges választás felmerülhet, bár leggyakrabban az egyes pixelek eltérésének négyzetösszegét használjuk. A block matching algoritmus ennek a négyzetes hibaösszegnek a minimumhelyét keresi meg, és ez a pozíció lesz az eredeti pixel párja.

Érdemes megjegyezni, hogy a block matching algoritmus az egyes pixelek diszparitását különkülön, egymástól függetlenül állapítja meg. Ez azért fontos, mert a diszparitás kép általában meglehetősen „sima” azaz a szomszédos pixelek diszparitás értékei egymáshoz közel vannak, és csak ritkán fordulnak elő nagy ugrások. Ez természetesen abból következik, hogy általában a valós háromdimenziós terek esetén is az objektumok térben összefüggők és csak a határaiknál fordul elő nagyobb térbeli ugrás. A block matching algoritmus azonban ezt nem veszi figyelembe, így a kapott diszparitás kép meglehetősen nagy zajjal terhelt.



14.3. ábra. *A BM algoritmus eredménye.*

14.3.2. SGBM

A valós terek tulajdonságait figyelembe véve nagymértékben lehet javítani a diszparitás kép minőségén. Ennek egy gyakran alkalmazott módszere az, ha a block matching hibakritériuma mellé felveszünk egy büntető tagot, ami a diszparitás simaságát írja elő. Ezt a megoldást alkalmazza a semi-global block matching (SGBM) módszer. Az SGBM módszer a közönséges block matching hibafüggvényét két újabb taggal egészíti ki. Az első tag során a vizsgált pixel egy adott környezetét megvizsgáljuk, és ha pixelek diszparitásai között pontosan 1 a különbség, akkor a hibafüggvényt egy P1 konstanssal növeljük. A második tag arról gondoskodik, hogy ha a különbség egynél nagyobb, akkor a hibafüggvényt egy P2 konstanssal büntetjük. A konstansok értékét tetszőlegesen választjuk a kapott eredményt figyelve, habár gyakran alkalmazott ökölszabály, hogy P2 értéke a P1 négyszerese.

$$E(D) = \sum_{p} E(p, D_p) + \sum_{q \in N(p)} P1T(|D_q - D_p| == 1) + \sum_{q \in N(p)} P2T(|D_q - D_p| > 1) \tag{14.1}$$

14.3.3. Belief propagation

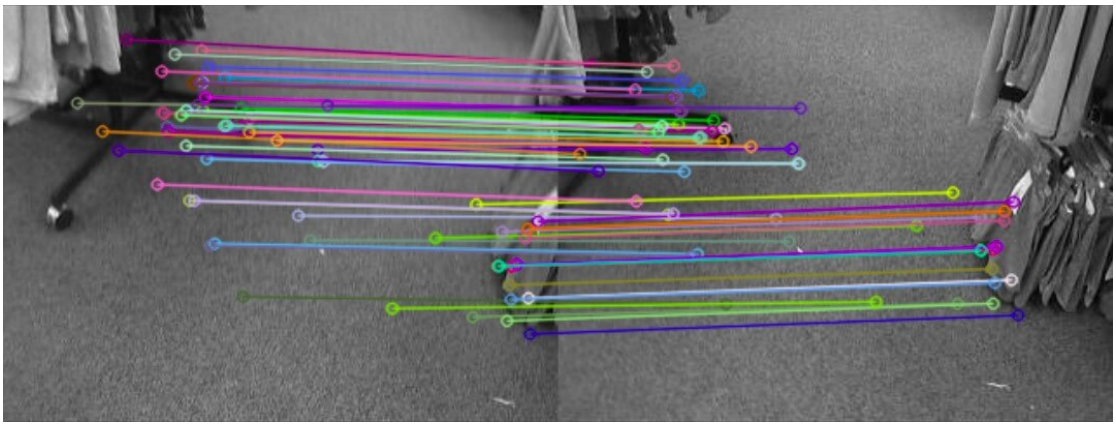
A diszparitás számolásának másik kiváló módszere a belief propagation (BP) módszer. Ennek az eljárásnak a lényege, hogy az egyes diszparitások értékét valószínűségi változóként értelmezi. Minden pixelnek ismeri a saját diszparitásának sűrűségfüggvényét, amelynek a maximumhelye az adott pixel „hite”. Ezt követően minden pixel üzenetet küld az összes szomszédjának: ezek az üzenetek voltaképpen a szomszédok diszparitására vonatkozó valószínűségi sűrűségek. Minden pixel úgy gondolja, hogy a szomszédjainak hozzá hasonló diszparitása van, így ez a sűrűség célszerűen egy normális eloszláshoz tartozik, ahol a várható érték az adott pixel hite.

Miután minden pixel kapott egy üzenetet az összes szomszédjától, a korábbi és a kapott sűrűségfüggvények felhasználásával minden pixel egy új sűrűségfüggvényt konstruál, az új hit pedig ennek a maximum helye lesz. Ezt a folyamatot addig ismételjük, amíg a diszparitás kép egy konszenzus állapothoz nem konvergál. Diszparitásképek esetére a konvergencia bizonyítottan garantált. Az algoritmus működése során a kezdeti sűrűségfüggvényt könnyedén generálhatjuk például a block matching módszer eredménye alapján: ez a metódus minden diszparitáshoz előállít egy hiba értéket, így a kezdeti valószínűségek lehetnek a hibákkal fordítottan arányosak.

Érdemes megjegyezni, hogy a belief propagation eljárás a pixelek diszparitása mellett bevezet még egyéb segédváltozókat is, amelyek az eltakarás és az objektumhatárok okozta diszparitás ugrásokat jelölik. A belief propagation algoritmus lényegesen jobb minőségű képet szolgáltat az SGBM módszerhez képest, azonban mindezt jelentős számításigény árán. A legtöbb rendszerben a BP eljárást érdemes inkább grafikus célhardver segítségével gyorsítani.

14.4. ábra. *A Belief Propagation algoritmus eredménye.*

Fontos még röviden megemlíteni azokat az eljárásokat, amelyek felhasználhatók diszparitás kép készítésére, azonban ennél általánosabb eljárások. Ezek közül az egyik az optikai áramlás algoritmus, amelynek különböző változatai korábban ismertetésre kerültek. A diszparitás képzéshez hasonlóan az optikai áramlásnak is léteznek sűrű, illetve ritka változatai, az optical flow eljárás azonban általános irányú elmozdulást is képes meghatározni. Amennyiben valamilyen okból kifolyólag nincs lehetőség a képek rektifikálására, akkor mindenképpen érdemes az optikai áramlást alkalmazni. Szintén lehetséges a korábban megismert lokális képjellemző leíró módszereket párosításra használni, habár ezeket a ritka eljárásokat gyakrabban szokták a kalibrációhoz szükséges pontpárok keresésére használni.



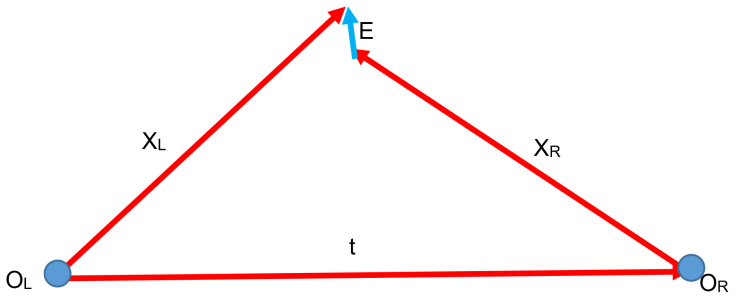
14.5. ábra. *SIFT képjellemzők párosítása.*

14.4. Rekonstrukció

A háromdimenziós rekonstrukció utolsó lépése a reprojekció, melynek során a kép pontjait visszavetítjük a háromdimenziós térbe. Erre a háromszögelés alapelvét használjuk fel.

14.4.1. Háromszögelés

A kalibráció során megismertük az egyes kamerák paramétereit, így minden egyes képponthoz elő tudjuk állítani azt a sugarat, amely mentén a képpontot képző fénysugár a kamerába beérkezett. Ezt természetesen az összes kamerára el tudjuk végezni, amin az adott képpont párját megtaláltuk. Továbbá, mivel ismerjük az egyes kamerák közötti transzformációkat, ezért az összes kamerát és sugarat transzformálhatjuk egy közös koordinátarendszerbe. Ebben a közös viszonyítási rendszerben az ugyanazon térbeli ponthoz tartozó vetítési sugarak az eredeti térbeli pont pozíciójában fogják egymást metszeni.



14.6. ábra. *A háromszögelés elve.*

Az imént ismertetett módszer a háromszögelés alapelve, amely különösen jól működik kétdimenziós problémák esetében. A háromdimenziós térben azonban abba a problémába ütközünk, hogy már

egészen apró pontatlanságok esetén is a két kamerából érkező sugarak könnyedén elkerülhetik egymást, így nem kapunk metszéspontot. Éppen ezért célszerűbb a háromdimenziós pont eredeti koordinátáit a legkisebb négyzetek módszerének segítségével meghatározni. Kezdetnek felírjuk a vetítés egyenletét, majd abból kifejezzünk a képpont u koordinátáját:

$$\begin{pmatrix} wu \\ wv \\ w \end{pmatrix} = \begin{pmatrix} p_{11}^T \\ p_{12}^T \\ p_{13}^T \end{pmatrix} \vec{X} \quad \rightarrow \quad u = \frac{wu}{w} = \frac{p_{11}^T \vec{X}}{p_{13}^T \vec{X}} \tag{14.2}$$

Ebben az egyenletben az u, v és a P összes eleme ismert, az X elemei azonban az ismeretlen térbeli koordináták. Ha a v koordinátát hasonló módon kifejezzük, majd a két kifejezésből egy egyenletrendszert írunk fel, akkor az alábbiakat kapjuk:

$$\begin{aligned} u p_{13}^T \vec{X} &= p_{11}^T \vec{X} \\ v p_{13}^T \vec{X} &= p_{12}^T \vec{X} \end{aligned} \tag{14.3}$$

Mely mátrixos formába rendezve:

$$\begin{pmatrix} p_{11}^T - u_1 p_{13}^T \\ p_{12}^T - v_1 p_{13}^T \end{pmatrix} \vec{X} = \vec{0} \tag{14.4}$$

Itt egy csupa ismert értékből álló mátrix szoroz balról egy csupa ismeretlen vektort, így ez az egyenlet a számunkra kényelmes, megoldható formában állt elő. A probléma azonban az, hogy az X vektornak négy ismeretlen eleme van (homogén koordináták), de csak két egyenlet áll rendelkezésünkre. Szerencsére van azonban még egy kameránk, amin ugyanennek az ismeretlen X pontnak a képe látszik, így a másik kamerából felírhatunk még két egyenletet. Ebben az esetben már négy egyenletünk van, így a megoldást a legkisebb négyzetek módszerével könnyedén megkaphatjuk. Fontos megjegyezni, hogy a fenti módszer kettőnél több kamera esetére triviálisan kiterjeszthető, hiszen minden kamera esetén újabb két egyenlet kerül az egyenletrendszerbe.

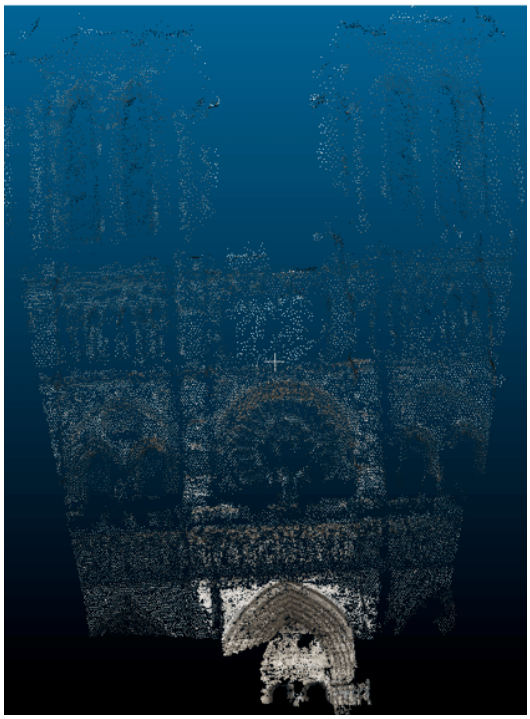
Számos térbeli számítógépes látás metódusokat tartalmazó függvénykönyvtár lehetőséget ad egy kifejezetten sztenderd sztereó elrendezésekre specializálódott reprojekciós művelet elvégzésére. Rektifikált képpárok esetén ugyanis nincs szükség legkisebb négyzetes becslésre, az eredeti térbeli koordináták egy egyszerű mátrixszorzás segítségével számolhatók. A rektifikáció kiszámítása során meghatározható egy Q mátrix, melynek segítségével az eredeti térbeli pozíciók a

$$\vec{X} = Q \begin{pmatrix} u \\ v \\ d \end{pmatrix} \tag{14.5}$$

formában adódnak, ahol u és v az X pont egyik képen megtalált pozíciója, d pedig a diszparitás értéke.

14.4.2. Metrikus rekonstrukció

Ennél a pontnál érdemes visszaemlékeznünk a háromdimenziós számítógépes látás területének eredeti motivációjára. A célunk az volt, hogy a kamera projekciója által torzított és információt vesztett képekből az eredeti háromdimenziós tér geometriáját rekonstruálni tudjuk. Tisztában kell azonban lennünk az itt ismertetett módszerek limitációival, és a működésük feltételeivel. Az egyik triviális, de mégis megemlítendő limitáció, hogy az egyszerű kétkamerás (sztereó) rekonstrukció sosem fog teljes 3D-s teret adni. Ennek oka egyszerűen az, hogy a sztereó elrendezésben a kamerák általában egymáshoz közel helyezkednek el, így a jelenet objektumainak csak az elülső felét látják. A kapott rekonstrukció hasonló lesz, az összes térbeli objektum hátulja hiányozni fog.



14.7. ábra. *A Notre Dame rekonstrukciója.*

Egy további fontos szempont a rekonstrukció metrikájának kérdése. Az alapvető cél az, hogy a kapott rekonstrukció méretei valamilyen ismert (lehetőleg SI) mértékegységben álljanak elő, azaz a rekonstrukció metrikus legyen. Érdemes ezért röviden a metrikus rekonstrukció feltételeit és azok hiányos teljesítésének következményeit tárgyalni. Egyszerűen fogalmazva a metrikus rekonstrukció feltétele, hogy a felhasznált kamerák belső paramétereit, illetve az elrendezés külső paramétereit maradéktalanul ismerjük.

Előfordulhat olyan eset (tipikusan mozgó kamerák esetén), hogy az elrendezés külső paramétereit csak hiányosan ismerjük (a két kép közti elmozdulás nagyságát nem tudjuk meghatározni). Ebben az esetben a rekonstrukció formahű lesz (geometriai torzításoktól mentes), az egyes objektumok mérete viszont nem lesz ismert. Ilyen esetekben a kapott koordináták egysége a két kamera közti elmozdulás szokott lenni. Érdemes megjegyezni, hogy ebben az esetben, ha bármilyen más módon az elmozdulás nagyságát, vagy esetleg a térben egy ismert objektum méretét becsülni tudjuk (sebességmérő, RGB-D szenzor), akkor a rekonstrukció metrikussá tehető. Természetesen, amennyiben erre nincs lehetőség a rekonstrukció még akkor is hasznos lehet, például járművek esetén az ütközések elkerüléséhez elegendő a távolságokat a jármű sebességének függvényében ismerni.

Előfordulhat azonban, hogy se a belső, se a külső paramétereket nem ismerjük, ebben az esetben a rekonstrukció művelete ugyan elvégezhető, de a kapott háromdimenziós jelenet egy ismeretlen projektív transzformáció erejéig lesz invariáns. Amint azt a fejezet bevezető részében láthattuk egy projektív transzformáció képes egyenesek párhuzamosságát megváltoztatni, vagy akár egy végtelen messzi pontot véges közelségbe hozni, így ilyen rekonstrukciók használata akár veszélyes is lehet.

14.5. Egy- és többkamerás módszerek

Az előző alfejezetben megismerkedtünk a sztereó rekonstrukció elvével, valamint annak hiányosságaival. Mivel a két kamerával készített rekonstrukció nem teljes, ezért gyakorta nevezik „2,5D” megoldásnak. Ezen felül érdemes még azt észrevenni, hogy a sztereó rekonstrukció során a térbeli pontok pozícióit az ahhoz szükséges lehető legkevesebb (kettő) mérésből határoztuk meg. Ebből az következik, hogy a kétkamerás rekonstrukció a lehető legzajosabb fajta, hiszen a mérések számának növelésével a becslés zaja elnyomható. Éppen ezért érdemes röviden a többkamerás rekonstrukció fajtáit is tárgyalni.

A rövid tárgyalás legfőbb oka, hogy az előző alfejezetben szinte mindent megismertünk, ami ezekhez a megoldásokhoz szükséges. Amennyiben több, fix kamerát használunk a teljes tér rekonstruálható, feltéve, hogy a kamerák a vizsgált térrészt minden irányból körbeveszik. Ebben az esetben maga a reprojekció az előző fejezetben ismert legkisebb négyzetes módszerrel nagy pontossággal számolható. Ezeket a megoldásokat leggyakrabban animációk és filmek készítésénél, illetve az orvosi képalkotásban használják, úgynevezett motion capture technológiák megvalósítására. Ekkor általában egy ember vagy állat mozgását rögzítik nagy pontossággal, három dimenzióban, hogy ezt az adatot aztán különböző célokra (animált karakterek készítése, diagnosztika) felhasználhassák.

14.5.1. SfM és SLAM

Valamivel érdekesebb az egy mozgó kamerát alkalmazó rekonstrukció esete. Ugyan itt valójában nehezen beszélhetünk többkamerás rekonstrukcióról, azonban mozgó kamerás esetben a rekonstrukciót általában számos felvételből készítik, így mégis a módszerek e csoportjába tartozik. A mozgókamerás rekonstrukciós módszereknek két gyakran használt elnevezése létezik: az SfM (az angol Structure from Motion – mozgásalapú struktúra) és a SLAM (az angol Simultaneous Localization and Mapping – egyidejű lokalizáció és térképalkotás). A két módszer közötti határvonal nem éles, bár a legtöbb irodalomban az SfM módszerek esetén a rekonstrukció készítése általában a felvétel készítése után offline, míg a SLAM esetében valós időben, online történik.

Az offline megoldások esetén a meglévő (nem feltétlenül sorban rendelkezésre álló) képek között megfeleltetéseket keresünk, majd a leggyakrabban előforduló pontokból, vagy a legtöbb megfeleltetést tartalmazó képpárból kezdjük a rekonstrukció építését. A megfeleltetéseket általában robusztus képjellemző detektáló módszerek (SIFT, SURF stb.) segítségével keressük.

14.8. ábra. *A Colosseum rekonstrukciója turista fotók alapján.*

Online megoldások esetén a képek a készítés sorrendjében érkeznek, és minden új kép esetén hozzáadunk a már meglévő rekonstrukcióhoz, miközben a kamera új pozícióját becsüljük. Ebben az esetben a becslést általában az előző képhez és állapothoz képest végezzük. A valósidejűségi követelmény miatt a párosításokat általában optikai áramlás, vagy hasonlóan gyors módszer segítségével végezzük. Egy tipikus SLAM algoritmus lépései a következők:

1. Jellemző detektálás
2. Az előző kép jellemzőivel párosítás
3. Kamera póz becslése
4. 3D pont koordináták becslése
5. Csoport igazítás (Bundle Adjustment)

Külön kiemelendő a csoport igazítás művelete, amely mind az offline, mind az online rekonstrukciós eljárások esetén előfordul. Ez a lépés meglehetősen hasonlít a belső kamerakalibrációs eljárások

utolsó, finomító lépésére, ugyanis ennél a lépésnél is egy iteratív numerikus optimalizáló eljárást használunk a geometriai hiba minimalizálására. Itt azonban nem a kamera belső paramétereit finomítjuk, hanem a háromdimenziós pontok koordinátáit és a kamera helyzetét leíró transzformációs mátrix elemeit.

A SLAM típusú algoritmusok egyik alapvető problémája a csúszás (drift) jelensége. Ez a jelenség abból adódik, hogy a kamera új pozícióját mindig az előző pozícióhoz képest becsüljük, így a kamera abszolút pozíciója relatív elmozdulások összességéből adódik. Mivel végtelen pontosan becsülni lehetetlenség, ezért az újabb és újabb becslések hibája egyre inkább halmozódik, és a kezdetekben kicsi hiba egyre nagyobb lesz, vagyis a becsült pozíció fokozatosan „elcsúszik" az igazitól.

Ennek a jelenségnek az orvoslására számos módszer létezik, melyek közül az első, hogy a relatív elmozdulást nem csak az előző, hanem korábbi képekhez képest is megbecsüljük, így mérsékelve a csúszás mértékét. Ezt természetesen nem lehet a végletekig művelni, hiszen a mozgó kamera egy idő után új területeket lát majd, és nem lesz átfedés a túl régi képkockákkal, így közös jellemzőket sem fogunk találni. Egy másik megoldás a hurokzár detektálás, melynek során érzékeljük, ha a kamera visszaért egy korábban meglátogatott pozíció közelébe, és ebben az esetben a korábbi pozícióban készített képkockához képest becslünk pozíciót. Továbbá általánosságban is lehetséges az a megoldás, hogy az előző képkockák mellett a már elkészült térképrészlethez képest is becsüljük a lokalizációt.

További Olvasnivaló

[1] R. Szeliski, *Computer Vision*. Springer London, 2011. DOI: 10.1007/978-1-84882-935-0. cím: https://doi.org/10.1007%2F978-1-84882-935-0.

[34] H. Hirschmuller, "Stereo Processing by Semiglobal Matching and Mutual Information", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30. évf., 2. sz., 328–341. old., 2008. febr. DOI: 10.1109/tpami.2007.1166. cím: https://doi.org/10.1109%2Ftpami.2007.1166.

[35] J. Sun, N.-N. Zheng és H.-Y. Shum, "Stereo matching using belief propagation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25. évf., 7. sz., 787–800. old., 2003. júl. DOI: 10.1109/tpami.2003.1206509. cím: https://doi.org/10.1109%2Ftpami.2003.1206509.

[36] T. Taketomi, H. Uchiyama és S. Ikeda, "Visual SLAM algorithms: a survey from 2010 to 2016", *IPSJ Transactions on Computer Vision and Applications*, 9. évf., 1. sz., 2017. jún. DOI: 10.1186/s41074-017-0027-2. cím: https://doi.org/10.1186%2Fs41074-017-0027-2.

[37] C. Wu, "Towards Linear-Time Incremental Structure from Motion", *2013 International Conference on 3D Vision*, IEEE, 2013. jún. DOI: 10.1109/3dv.2013.25. cím: https://doi.org/10.1109%2F3dv.2013.25.

15. fejezet

3D Feldolgozás

15.1. Bevezetés

A korábbi fejezetek során részletesen tárgyaltuk a háromdimenziós rekonstrukció elvégzésének lehetséges módjait. Nyilvánvaló azonban, hogy a feladat ennél a pontnál még nem ér véget, ugyanis a rekonstrukciót további feldolgozás céljából hoztuk létre. Éppen ezért szükséges a térbeli struktúrák feldolgozásának módszeriről röviden beszélnünk. A rekonstrukció során előálló adathalmaz feldolgozásának módszerei ugyanis valamelyest hasonlóak a számítógépes látás korábban ismertetett algoritmusaihoz, azonban néhány alapvető tulajdonságuk lényegesen különbözik.

A háromdimenziós struktúrák feldolgozásának alapfeladatai voltaképpen ugyanazok, mint a számítógépes látás esetében. Az előállt adathalmaz zajokkal és hibákkal terhelt, tehát szükség lesz szűrések végrehajtására. Ezt követően valamilyen módszerrel el kell különítenünk, vagy fel kell ismernünk a tér számunkra releváns részét. Ez utóbbit végezhetjük valamilyen szegmentációs, vagy lokális jellemzőkre épülő felismerési módszer segítségével.

15.2. 3D Struktúra reprezentációja

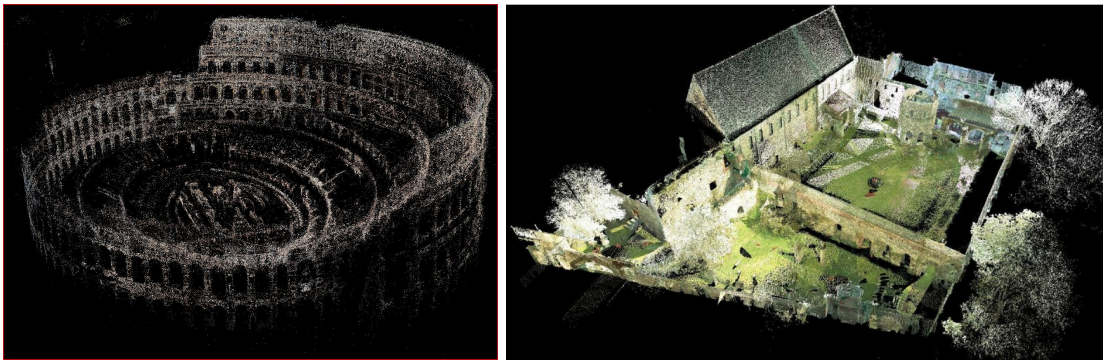
Mielőtt azonban belekezdnénk az egyes módszerek ismertetésébe, egy alapvető kérdést szükséges még tisztáznunk. Az ugyanis nem nyilvánvaló, hogy a térbeli struktúrát milyen formában reprezentáljuk. A hagyományos számítógépes látás egy kétdimenziós képet egy kétdimenziós rácson tárol, ahol a rács minden eleme egy pixel értéke. Magától értetődő megoldás volna egy térbeli struktúrát úgynevezett voxelek (a volumetrikus és a pixel szó összemosása) háromdimenziós rácsaként tárolni, ahol minden voxelhez egy kicsi kocka alakú térfogatrész tartozik, a voxel értéke pedig a hozzá tartozó térrészben található objektumtól függ.

Az előbb ismertetett megoldás azonban egy meglehetősen rossz ötlet több szempontból is, melyek közül az első a dimenzionalitás átka. Képzeljük el, hogy egy átlagos felbontású kép mindkét dimenziójában 1-4000 pixelt tartalmaz, azaz összesen néhány millió pixelből áll, vagyis a memóriában néhány megabájt méretet foglalnak. A térbeli struktúráknak azonban eggyel több dimenziója van, tehát hasonló felbontás mellett a voxelek száma néhány milliárd lesz, vagyis a szükséges memória a gigabájtos tartományban mozogna. Arról nem is beszélve, hogy a rekonstrukciót gyakran számos képből állítjuk elő, így míg egyetlen kép a teljes térnek csak egy kis részét látja, a rekonstrukción az egész szerepelni fog, ahhoz pedig a képeknél megszokottnál nagyobb felbontásra volna szükségünk.

További probléma a voxelrácsos megoldással, hogy rendkívül pazarló. Míg általában egy kép minden pixelére esik fény, szemben a háromdimenziós terek meglehetősen üresek. Bíztatom az olvasót, hogy nézzen körül; amennyiben nem egy raktárhelységben tartózkodik, akkor a helység teljes térfogatának a 90-95 százaléka valószínűleg üres. Ehhez vegyük hozzá, hogy természetüknél fogva a háromdimenziós rekonstrukciók csak az objektumok felületét tartalmazzák (a kamerák

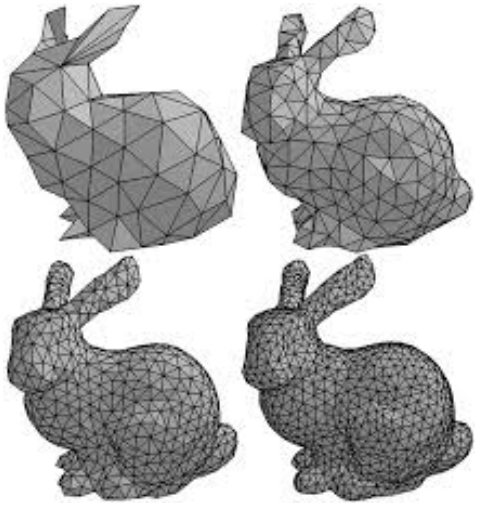
nem látnak be az objektumok belsejébe), így egy átlagos rekonstrukció térfogatának nagyjából 1 százalékában található bármi is.

Éppen ezért a háromdimenziós reprezentációk alapvető tároló struktúrája a pontfelhő, amely egyszerűen az objektumokat alkotó háromdimenziós pontok (általában rendezetlen) listája. A fent tárgyaltak fényében ez a módszer takarékosabb és pontosabb is, mint a rács használata. Érdemes megjegyezni, hogy a pontfelhőkben lévő pontok a pozíciójukon felül egyéb adatokat is gyakran tárolnak a gyakorlatban. Ezek közül az egyik leggyakoribb eset, amikor minden ponthoz hozzárendeljük annak a pixelnek a színét, amely az egyes képeken az adott ponthoz tartozik. Szintén gyakori, hogy a pontok alkotta felületek normális vektorát kiszámítjuk, és minden ponthoz hozzárendeljük.



15.1. ábra. *Bináris (bal) és színes (jobb) pontfelhők.*

Érdemes még megemlíteni, hogy a pontfelhőn kívül léteznek még magasabb szintű leírási módszerek háromdimenziós objektumok esetére. Különböző struktúrákat lehetséges különböző primitív formákkal (gömb, sík, henger, kúp stb.), vagy paraméteres felületekkel és görbékkel közelíteni, így potenciálisan nagy ponthalmazokat csupán néhány számmal leírni. Szintén elterjedt megoldás a háromszögháló (mesh) használata, ahol komplex felületeket háromszög alakú építőelemekkel közelítenek. Ez utóbbi módszer elsősorban a számítógépes grafikában elterjedt.

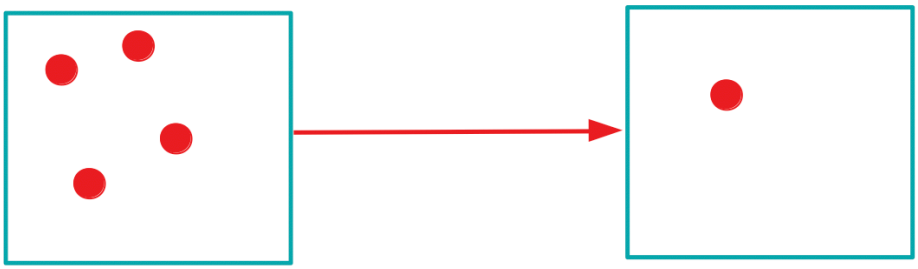


15.2. ábra. *Egy mesh segítségével reprezentált felület eltérő részletességgel.*

15.3. Szűrési módszerek

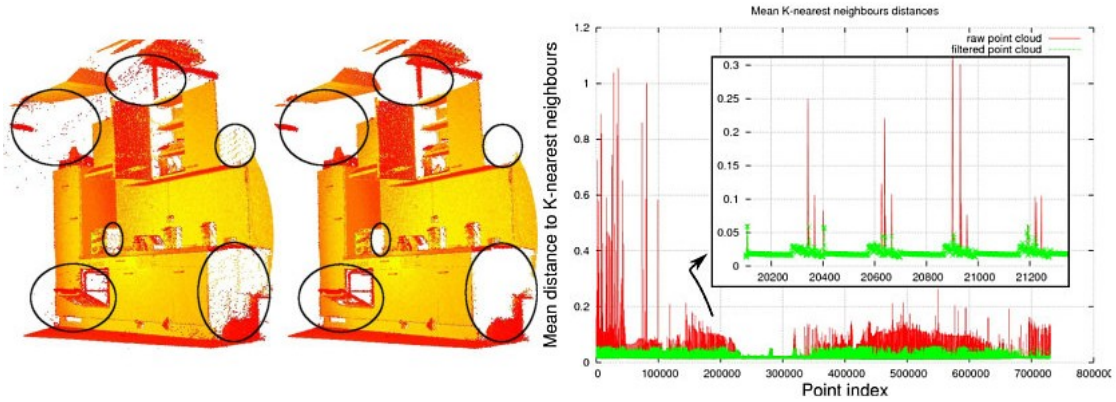
Ahogy a számítógépes látásnak, így a pontfelhők feldolgozásának is alapvető lépése a szűrés és javítás. Háromdimenziós pontfelhők esetén a legegyszerűbb megoldás erre az úgynevezett doboz, vagy voxel szűrő. A szűrő működése analóg a képeknél megismert átlagoló szűrőhöz: a doboz szűrő

egy háromdimenziós ablakkal végigpásztázza a pontfelhőt, és minden lépésben a doboz területébe eső pontokat az átlagukkal helyettesíti. A doboz szűrő kiváló véletlen pozíciózajok kiszűrésére, valamint a pontfelhő alulmintavételezésére. Előfordulhat ugyanis, hogy a rekonstrukció, vagy az illesztések hibái miatt ugyanaz a térbeli pont többször is szerepel a pontfelhőben minimálisan eltérő koordináta értékekkel. Ilyen esetekben újabb felvételek hozzáadása a pontfelhő méretét a végtelenségig növelhetné, a doboz szűrő segítségével azonban ezt kordában tudjuk tartani.



15.3. ábra. *A doboz szűrő szűrési elve.*

Léteznek azonban olyan hibás pontok a pontfelhőkben, amikkel szemben a doboz szűrő tehetetlen. A rekonstrukció során gyakran előfordul, hogy például helytelen párosítás miatt egy-egy pont a felhőben teljesen hibás, és minden más ponttól messze, az egész felhőből kívülre esik. Ezeket a pontokat outlier-nek nevezzük, és könnyen belátható, hogy nem lehet őket a közeli szomszédjaival összeátlagolni, mivel nincsenek közeli szomszédjai. Ezt azonban könnyedén ki lehet használni a kiszűrésükre: egyszerűen keressük meg minden ponthoz a k darab legközelebbi szomszédját és számoljuk ki a tőlük vett átlagos távolságot. Azok a pontok, ahol ez az átlagos távolság jelentősen eltér a többi ponttól outliernek minősülnek.



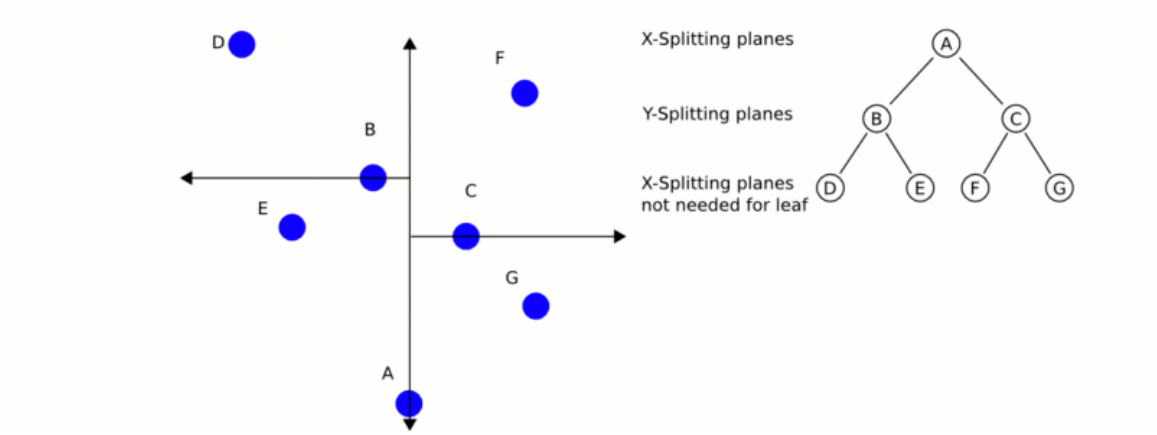
15.4. ábra. *Outlier pontok (bal) és a szűrésükhöz használható hisztogram (jobb).*

15.3.1. Kd-fa reprezentáció

Érdemes észrevenni, hogy mindkét fajta szűréshez szükségünk van az egyes pontok közeli, vagy legközelebbi szomszédjainak megkeresésére. Képek esetén ez egyszerű, hiszen egy pixel legközelebbi szomszédjai ott vannak mellette. Pontfelhők esetében azonban a pontok nincsenek feltétlenül bármilyen logikus sorrendben, így a legközelebbi szomszéd megtalálásához N darab pontot végig kell ellenőriznünk, ahol N a pontfelhőben lévő pontok száma.

Ezt a problémát próbálja orvosolni az úgynevezett kd-Fa struktúra. A kd-fa a pontfelhő pontjait egy gráfban helyezi el, ahol egy gyökér csomópont található, és minden csomópontnak két gyereke van (más szóval a kd-fa egy bináris fa). A fa konstruálásának kezdetekor kiválasztunk egy pontot gyökérpontnak, majd egy tetszőleges dimenzió mentén egy síkkal kettéválasztjuk a pontfelhőt (ennek a síknak tartalmaznia kell a pontot). Ezt követően megkeressük a síkhoz legközelebb eső

pontot mind a két részpontfelhőből, és ezek a pontok lesznek a kezdeti pont két gyereke. Ezt követően a két gyerekpontra megismételjük a gyökérpontra elvégzett műveleteket (pontot tartalmazó síkkal elmetszés, új gyerekpontok keresése) ügyelve arra, hogy a gyerekeknél használt metsző sík mindig merőleges legyen a szülőnél használt síkra.



15.5. ábra. *A kd-fa konstruálási szabálya.*

A kd-fa használatának hatalmas előnye, hogy a megkonstruálása után (amit csak egyszer kell megcsinálni), ha legközelebbi szomszédokat kell keresni egy pontfelhőben, akkor a bináris fa struktúrájából adódóan minden lépésben a pontfelhő egyik felét teljesen ki tudjuk zárni a keresésből. Ez azt jelenti, hogy N lépés helyett annak kettes alapú logaritmusával lesz arányos a keresési idő, ami hatalmas gyorsulást jelent, különösképpen nagy, több millió pontot tartalmazó felhők esetében.

15.4. Szegmentáció

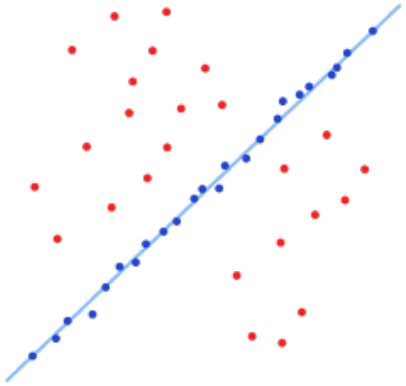
A szegmentáció a pontfelhők feldolgozásának egy alapvető fontosságú feladata, ugyanis ennek segítségével képesek lehetünk elválasztani az egyes objektumokat egymástól, illetve egyéb, számunkra érdektelen háttérobjektumoktól. A pontfelhők feldolgozásában nagyon gyakori feldolgozási lépés a padló/föld eltávolítása, a legtöbb esetben ugyanis ez egy olyan háttérobjektum, ami az összes releváns objektumot összeköti.

A negyedik előadáson megismerkedtünk számos képeken használható szegmentálási módszerrel, többek között a régió alapú módszerekkel, mint a régiónövesztés és -szeletelés módszere, a klaszterezés alapú eljárásokkal, mint a k-means és a MoG, valamint a gráfvágásra alapuló megoldásokkal. Ezek a szegmentálási módszerek pontfelhők esetén szinte pontosan ugyanúgy működnek, mint képeknél. Az egyetlen apróbb különbség, hogy amíg képek esetén az egyes pontok közti hasonlóság mércéje általában a pixel értékek hasonlósága volt, pontfelhők esetében gyakrabban használnak a térbeli összetartozásra alapuló mércéket.

15.4.1. RANSAC

Létezik azonban egy algoritmus, amelyet ritkán használnak képekre, pontfelhők szegmentálása esetében azonban az egyik legnépszerűbb eljárásnak számít. Ez az algoritmus a RANSAC (Random Sample Consensus – véletlen minták konszenzusa) névre hallgat. A RANSAC alapvetően egy univerzális paraméterbecslési eljárás, amelyet a pontfelhők szegmentálásán kívül még számos helyen használnak: többek között kamerakalibrációra is.

Az algoritmus alapelve rendkívül egyszerű: egy ponthalmazból véletlenszerűen pontokat kiválasztva egy jelöltet készít a megoldásra, majd ezt újabb véletlen választásokkal ismételve nagy számú véletlen jelöltet állít elő. Ezt követően megvizsgálja, hogy az egyes jelöltekre a teljes ponthalmazból hány pont illeszkedik rá. Az egy adott jelöltre illeszkedő pontokat inlier-nek hívjuk. A

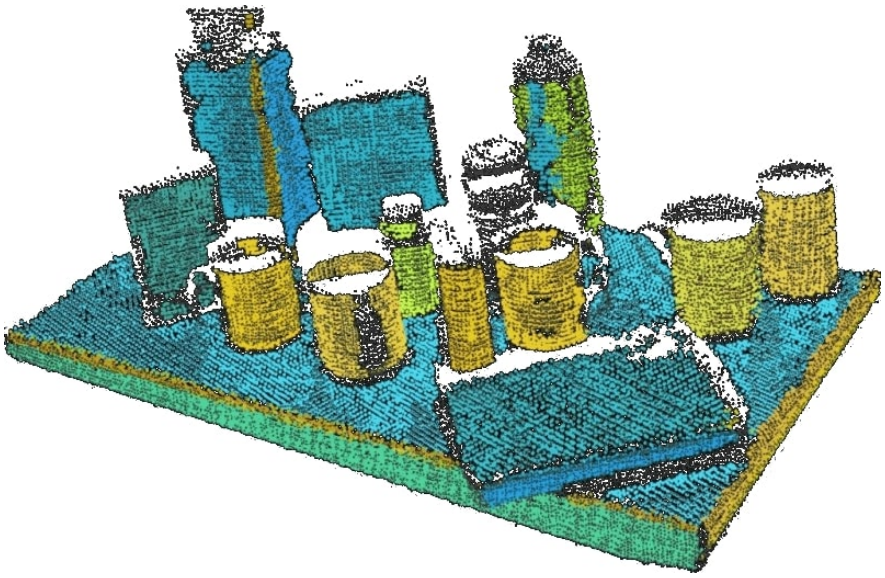


15.6. ábra. *Inlier és outlier pontok egyenes illesztésekor.*

RANSAC algoritmus minden jelöltre összeszámolja az inliereket, és eredményként visszaadja a legtöbb inlierrel rendelkező jelöltet.

Pontfelhők esetén a RANSAC eljárást tipikusan primitív paraméteres formák (síkok, hengerek stb.) keresésére használják. Ekkor a RANSAC algoritmus véletlenszerűen kiválaszt néhány pontot a felhőből, és meghatározza azt a primitív formát, amely ráillik ezekre a pontokra. Számos ilyen jelölt elkészítése után megvizsgálja, hogy a teljes pontfelhőből hány pont illik rá az egyes primitív formák által meghatározott felületekre. Érdemes észrevenni, hogy mivel a RANSAC algoritmus paraméteres formákat keres, ezért ezeket a primitíveket teljesen skála- és elforgatásinvariáns módon képes megtalálni.

Érdemes megjegyezni, hogy a módszernek számos variációja létezik, amelyek általában a véletlen pontok kiválasztásánál, illetve a kiértékelésnél eszközölnek módosításokat. Pontfelhő szegmentálás esetében például célszerű a teljesen véletlen mintavételezést egy lokálisan véletlen megoldásra cserélni. Ez azt jelenti, hogy egymáshoz relatíve közeli véletlen pontokból konstruáljuk a jelölteket, hiszen a geometriai alakzatok alapvetően lokális jelenségek, így közeli pontok nagyobb eséllyel tartoznak ugyanahhoz a formához. A kiértékelés során szintén érdemes egy lokális környezeten belül maradni hasonló megfontolásokból. Az algoritmussal kapcsolatban általánosságban is elmondható, hogy a mintavételezés és kiértékelés módszereit érdemes az adott feladatra szabni.



15.7. ábra. *RANSAC alakfelismerés pontfelhőn. Ez a modell sík és henger alakzatokat igyekezett szegmentálni.*

Mint már említettem a RANSAC eljárás általánosan használható paraméterbecslési feladatokra, ilyen szempontból a legkisebb négyzetes becslés módszerének a riválisa. A RANSAC hatalmas előnye azonban, hogy az outlier pontokra nagyrészt érzéketlen. Míg a legkisebb négyzetes becslés a véletlen hibákat jól kezeli, ha azonban a teljesen rossz outlier pontok aránya 50% fölé emelkedik, akkor a becslés eredménye már teljesen rossz lesz. Az algoritmus hátránya azonban, hogy véletlenszerű a működése, így nem garantálható, hogy megtalálja a helyes megoldást. A valószínűség növeléséhez a jelöltek számát kell tovább növelnünk, ami lassú működést eredményez. Elmondható, hogy a RANSAC tipikusan nem valósidejű algoritmus.

15.5. Objektumfelismerés

Különböző objektumok és osztályok felismerése, valamint detektálása a számítógépes látás egyik legfontosabb feladata. Éppen ezért érdemes külön fejezetet szentelni a pontfelhőkben elvégzett objektumfelismerésnek. Maga a felismerés és detektálás menete a képek esetéhez rendkívül hasonló. Első lépésben egy referenciaobjektumot veszünk, majd kiszámoljuk az objektum valamilyen, a felismeréshez jól használható jellemzőit. Ezt követően a pontfelhő objektumaira is kiszámoljuk ugyanezen jellemzőket, majd a jellemzők között párosítást végzünk. Utolsó lépésként a párosítások alapján megbecsüljük a referencia és a detektált objektum közti transzformációt.

Ennek az eljárássorozatnak a legfontosabb lépése a jellemzők meghatározása, így a jelen fejezetnek a fő témája az ilyen módszerek bemutatása lesz. A pontfelhőkben számolt jellemzőknek alapvetően két fajtája létezik: lokális jellemzők, amelyek egyes pontokat írnak le, illetve globális jellemzők, amelyek egész objektumokat, vagy szegmenseket jellemeznek. Érdemes megjegyezni, hogy a jellemzőkkel szemben a képi esethez hasonló megkötéseink vannak: legyenek elég egyediek az objektumfelismerés hatékony elvégzéséhez, és legyenek robusztusak a különböző transzformációkra.

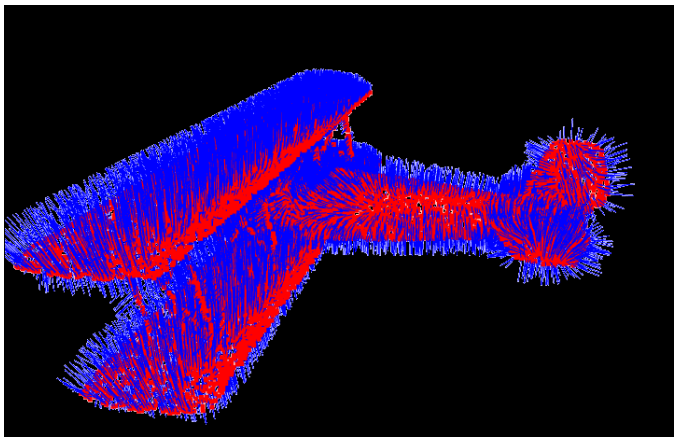
15.5.1. Lokális leírók

Az egyik leggyakrabban használt lokális jellemző pontfelhők esetében a felületi normális vektor. Ez egy olyan jellemző, amely szinte minden ponthoz számolható és a vizsgált pont környezetében lévő pontok által alkotott felület irányultságát írja le. Számolása meglehetősen egyszerű: a felület ugyanis egy lokálisan kétdimenziós ponthalmaz, ami azt jelenti, hogy a vizsgált pont közeli szomszédjai két irányba szóródnak, az erre merőleges harmadik irányban viszont nem. Így, ha kiszámoljuk a vizsgált ponthalmaz kovarianciamátrixát, annak legkisebb sajátértékéhez tartozó sajátvektora meg fogja határozni azt az irányt, amerre a ponthalmaz a legkevésbé szóródik: ez lesz a normális iránya. Érdemes megjegyezni, hogy az így kapott két lehetséges irány (a kapott sajátvektor és annak mínusz egyszerese) közül úgy döntünk, hogy a normális konvenció szerint mindig a nézőpont felé mutat.

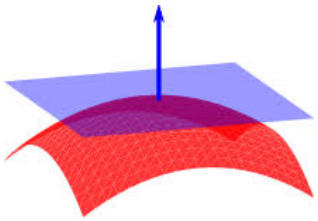
Bár a normális vektorok számítása egyszerű, önmagukban nem elég egyediek ahhoz, hogy pontokat pusztán ezek alapján párosíthassunk. Felhasználhatók azonban összetettebb jellemzők konstruálására, ahogy azt a pontjellemző hisztogram (PFH – Point Feature Histogram) módszer is teszi. Ez az eljárás minden leírandó ponthoz megkeresi annak n legközelebbi szomszédját. Ezt követően ebből az n+1 pontból minden lehetséges módon pontpárokat készít, és minden pontpárhoz kiszámol néhány jellemzőt. Ezek a jellemzők lehetnek a pontok távolsága, a két pont normálisai által bezárt szög, valamint a két pontot összekötő szakasz és a normálisok által bezárt szögek. Miután ezeket a jellemzőket minden pontpárra meghatározzuk, ezekből hisztogramokat készítünk, és ezeket használjuk lokális leíró gyanánt. Érdemes ennél a pontnál visszaemlékezni a képek esetében alkalmazott SIFT eljárásra, amely nagyon hasonló elven működött.

15.5.2. Globális leírók

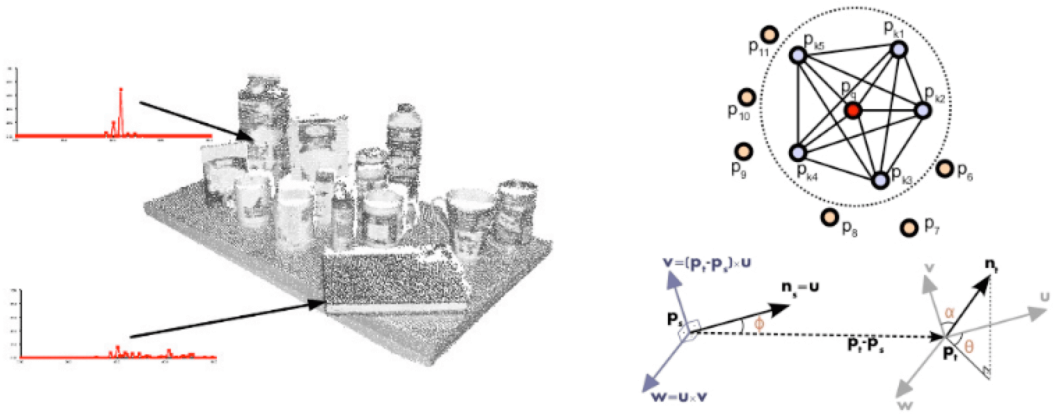
Globális, egy egész objektumot leíró jellemzőre rendkívül jó példa a GASD (Globally Aligned Spatial Distribution – globálisan illesztett térbeli eloszlás) módszere. Ennek a módszernek a lényege,



15.8. ábra. *Egy pontfelhő normálisai.*



15.9. ábra. *Egy felület és a normális iránya.*

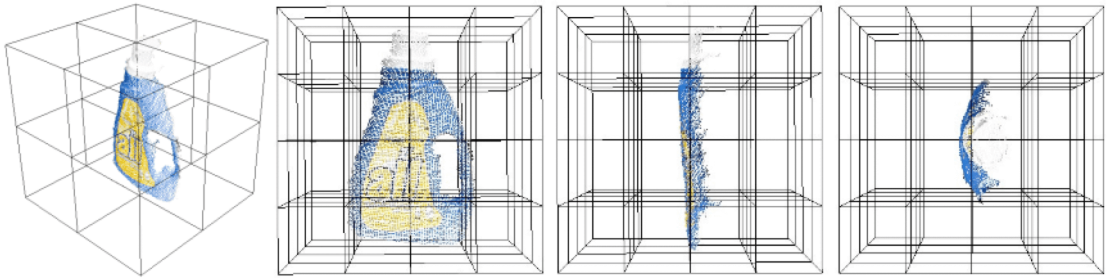


15.10. ábra. *A PFH során a szomszédos pontok között használt jellemzők (jobb) és néhány példa hisztogram (bal).*

hogy az objektumot alkotó pontok kovarianciamátrixának sajátértékeit és sajátvektorait kiszámolja, majd a pontfelhőt úgy forgatja el, hogy a legnagyobb sajátértékhez tartozó vektor az x, a legkisebb pedig a z irányba álljon. Ezen a módon az összes objektumot egy konzisztens orientációba forgatjuk. Ezt követően a teret felosztjuk egy $M \times M \times M$ felbontású rácsra, és minden cellában megszámoljuk a pontok számát, és ezekből egy hisztogramot készítünk. Ezt a hisztogramot használjuk az objektum leírójaként. Érdemes megjegyezni, hogy az itt bemutatott három jellemző generálási módszeren felül még számos egyéb módszer létezik mind lokális, mint globális leírók készítésére.

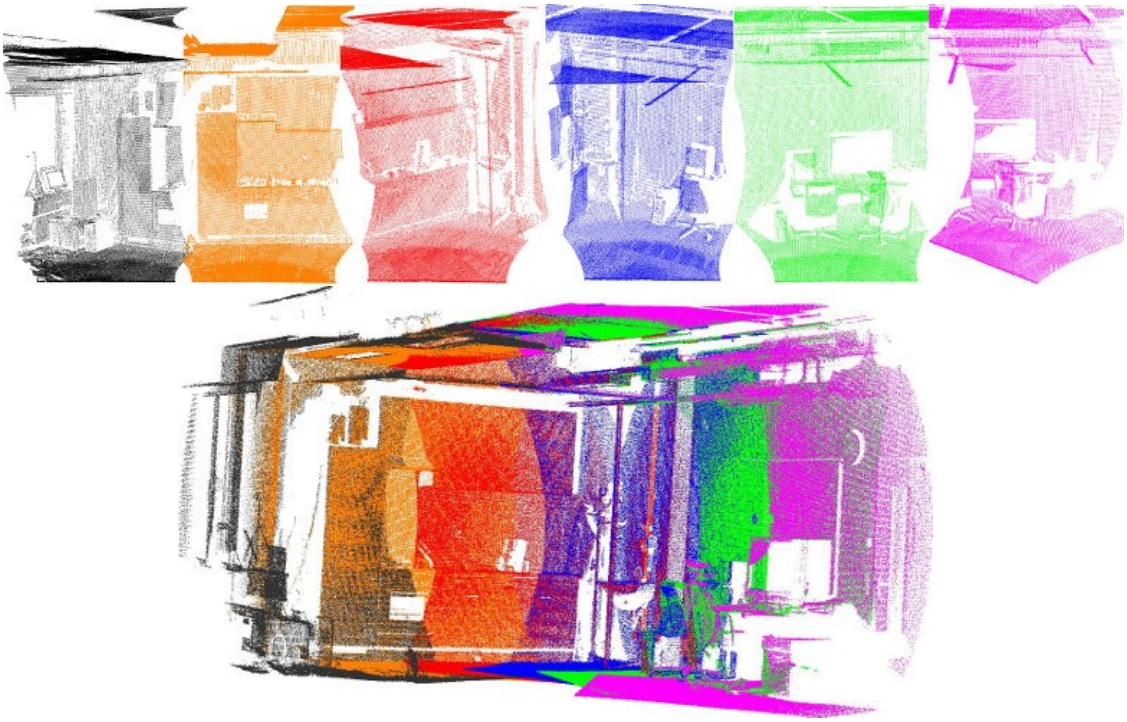
15.5.3. Regisztráció

Az objektumok felismerésén és detektálásán felül a lokális leíróknak van még egy fontos felhasználása: ez a regisztráció művelete. A regisztráció feladata során ugyanarról a térről kettő vagy



15.11. ábra. *A globálisan leírandó objektum (bal) és a szóródási irányok alapján beforgatott változatai.*

több egymással részleges átfedésben lévő pontfelhő részlet áll rendelkezésre. A művelet lényege, hogy ezeket a részleteket illesszük össze egyetlen a teljes teret leíró pontfelhővé. A regisztráció végrehajtásakor az átfedések detektálására gyakran alkalmaznak lokális jellemzőket.



15.12. ábra. *Pontfelhő részletek regisztráció előtt (felül) és után (alul).*

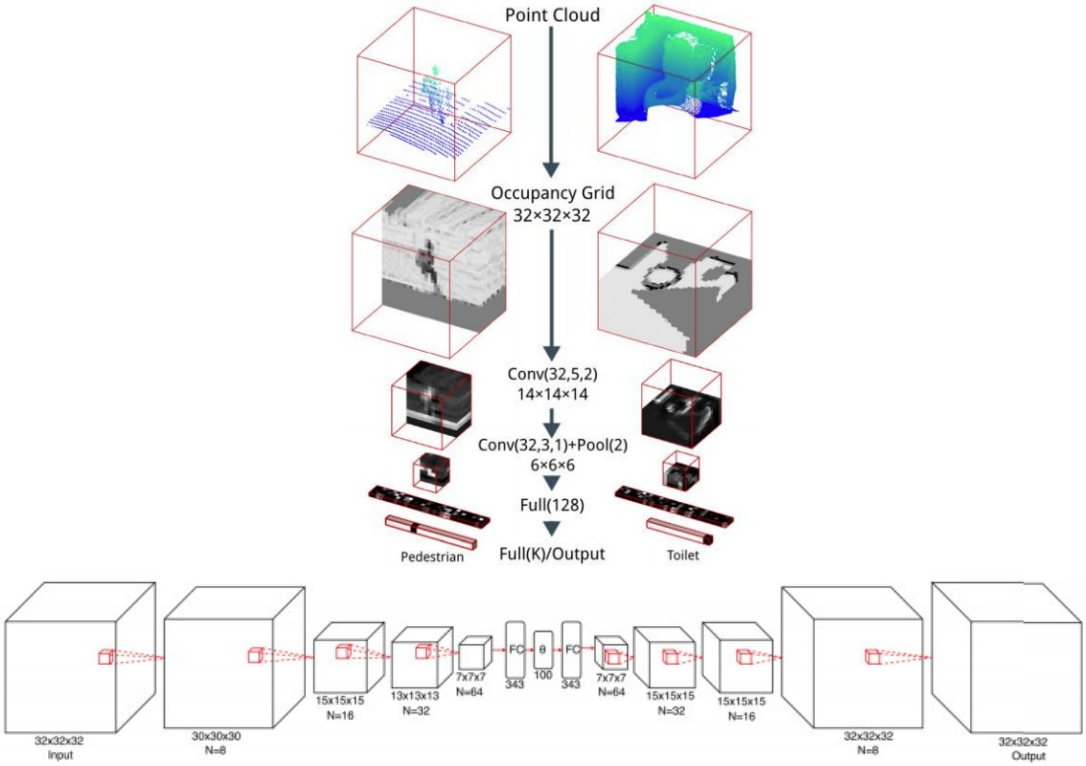
15.6. 3D Deep Learning

A deep learning módszereinek alkalmazása alapvető nehézségekbe ütközik a 3D feldolgozás esetén, ami jelentősen megnehezíti ezen módszerek használatát.

15.6.1. Voxel hálók

Adja ugyanis magát az ötlet, hogy használjunk 2D konvolúciós hálók helyett 3D hálót voxeles adatokon. Azonban ahogy azt az előadás elején megbeszéltünk, ez rendkívül problémás, mivel a tárolásnak nagy memóriaigénye van, ráadásul pazarló is. Ez a deep learning esetében különösen nagy probléma, mivel a mély neurális hálók már képek esetén is hatalmas memóriaigénnyel rendelkeznek, ami az egyik legfőbb szűk keresztmetszetet jelenti a modern GPU-k alkalmazásánál.

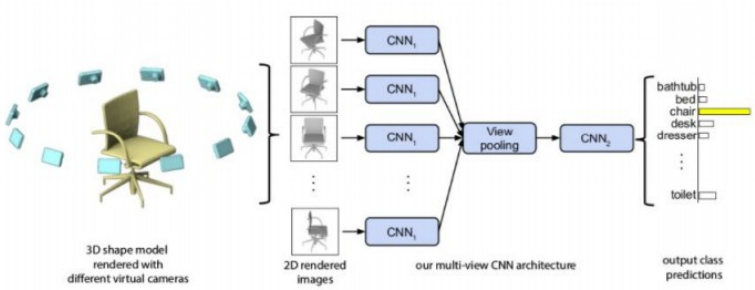
Ennek ellenére léteznek voxel neurális hálók, ezek azonban elég kicsi felbontáson működnek csak, így a teljesítményük is limitált.



15.13. ábra. *Egy voxel alapú osztályozó (felül) és egy szegmentáló (alul) neurális háló.*

15.6.2. Projekción alapuló hálók

Léteznek még neurális hálók, amik a 3D pontfelhőből véletlenszerűen generált nézőpontokból 2D vetületeket állítanak elő a vetítés egyenlete alapján, majd ezt egy hagyományos 2D konvolúciós háló segítségével osztályozzák. Ezzel visszavezethető a 3D osztályozás művelete 2D osztályozásra, azonban ez is jelentős lassulással járul, hiszen több képet kell végigfuttatni ugyanazon a hálón. Továbbá ezen módszerek igen nehezen terjeszthetők ki további problémák (detektálás, szegmentálás) megoldására.



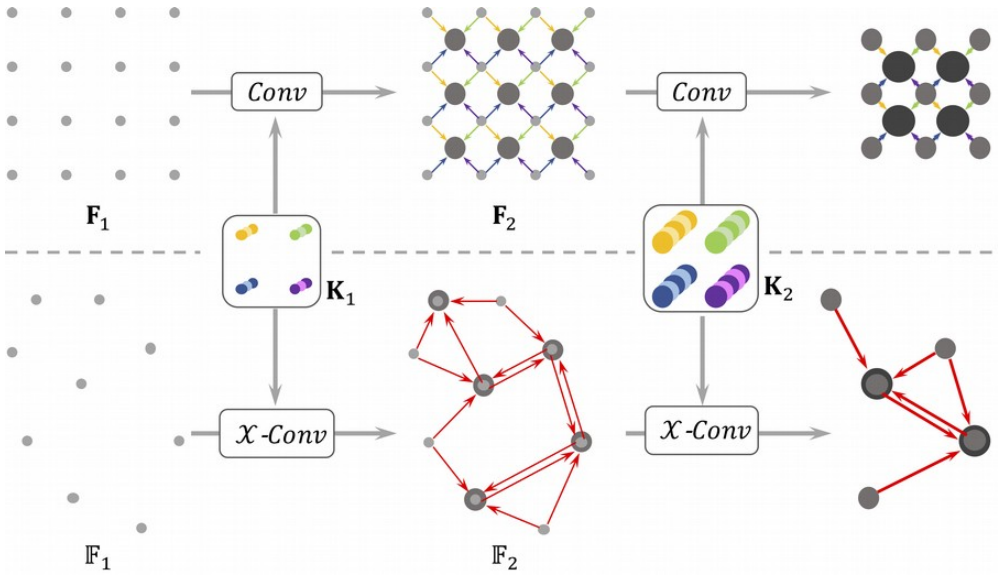
15.14. ábra. *A Projekció alapú mély tanulás sémája.*

15.6.3. Pontfelhő hálók

Ha viszont megkíséreljük valamelyik kézenfekvőbb reprezentációs módszert alkalmazni, akkor hamar szembesülhetünk azzal, hogy ezek kevésbé foghatók meg jól konvolúciós hálók segítségével.

Pontfelhők esetében például azzal a problémával szembesülünk, hogy a neurális hálók alapvetően rendezett adathalmazokon tudnak jól működni. Ha a pontfelhő felsorolásában megcserélek két pontot, akkor még mindig ugyanazt a pontfelhőt írják le, egy hagyományos neurális háló viszont nem ugyanazt fogja végrehajtani.

Erre egy lehetséges megoldás, ha egy transzformáció segítségével a pontokat négyzetrácsos elrendezésbe hozzuk, majd ezt követően végezzük el a konvolúciót. Ez úgy képzelhető el, hogy minden pontnak megkeressük az egyes irányokba található legközelebbi szomszédját, majd ezeket tekintjük közvetlen szomszédnak a rácson. Az így elvégzett konvolúciót χ-Konvolúciónak nevezzük. A konvolúciót itt tovább módosíthatjuk úgy, hogy ne csak a szomszédos pontok/jellemzők értékeit, hanem azok távolságát is figyelembe vegye. Ezt a megoldást alkalmazza az úgynevezett Point-CNN architektúra.



15.15. ábra. *A hagyományos konvolúció (felül) és a χ-konvolúció (alul).*

További Olvasnivaló

[1] R. Szeliski, *Computer Vision*. Springer London, 2011. DOI: 10.1007/978-1-84882-935-0. cím: https://doi.org/10.1007%2F978-1-84882-935-0.

[38] J. L. Bentley, "Multidimensional binary search trees used for associative searching", *Communications of the ACM*, 18. évf., 9. sz., 509–517. old., 1975. szept. DOI: 10.1145/361002.361007. cím: https://doi.org/10.1145%2F361002.361007.

[39] R. Schnabel, R. Wahl és R. Klein, "Efficient RANSAC for Point-Cloud Shape Detection", *Computer Graphics Forum*, 26. évf., 2. sz., 214–226. old., 2007. jún. DOI: 10.1111/j.1467-8659.2007.01016.x. cím: https://doi.org/10.1111%2Fj.1467-8659.2007.01016.x.

[40] R. B. Rusu, N. Blodow és M. Beetz, "Fast Point Feature Histograms (FPFH) for 3D registration", *2009 IEEE International Conference on Robotics and Automation*, IEEE, 2009. máj. DOI: 10.1109/robot.2009.5152473. cím: https://doi.org/10.1109%2Frobot.2009.5152473.

[41] J. P. S. do Monte Lima és V. Teichrieb, "An Efficient Global Point Cloud Descriptor for Object Recognition and Pose Estimation", *2016 29th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, IEEE, 2016. okt. DOI: 10.1109/sibgrapi.2016.017. cím: https://doi.org/10.1109%2Fsibgrapi.2016.017.

[42] Y. Zhou és O. Tuzel, "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection", *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, IEEE, 2018. jún. DOI: 10.1109/cvpr.2018.00472. cím: https://doi.org/10.1109%2Fcvpr.2018.00472.

[43] H. Su, S. Maji, E. Kalogerakis és E. Learned-Miller, "Multi-view Convolutional Neural Networks for 3D Shape Recognition", *2015 IEEE International Conference on Computer Vision (ICCV)*, IEEE, 2015. dec. DOI: 10.1109/iccv.2015.114. cím: https://doi.org/10.1109%2Ficcv.2015.114.

[44] Y. Li, R. Bu, M. Sun, W. Wu, X. Di és B. Chen, *PointCNN: Convolution On X-Transformed Points*, 2018. eprint: 1801.07791. cím: http://www.arxiv.org/abs/1801.07791.

IV. rész

Valósidejű Látás

16. fejezet

Hardverek

16.1. Bevezetés

Mivel a számítógépes látás során általában hatalmas adatmennyiség feldolgozását kell valós időben megvalósítani, ezért célszerű külön tárgyalni az egyes algoritmusok gyorsításának lehetőségeit. A folyamatosan növekvő processzorteljesítmény ugyan a fejlesztők oldalán áll, azonban általában több év szükséges a számottevő növekedés eléréséhez, amire a fejlesztés során nem lehet várni. Éppen ezért érdemes az algoritmusok gyorsításánál más megoldások felé tekinteni. Alapvetően a gyorsításnak három fő paradigmája létezik:

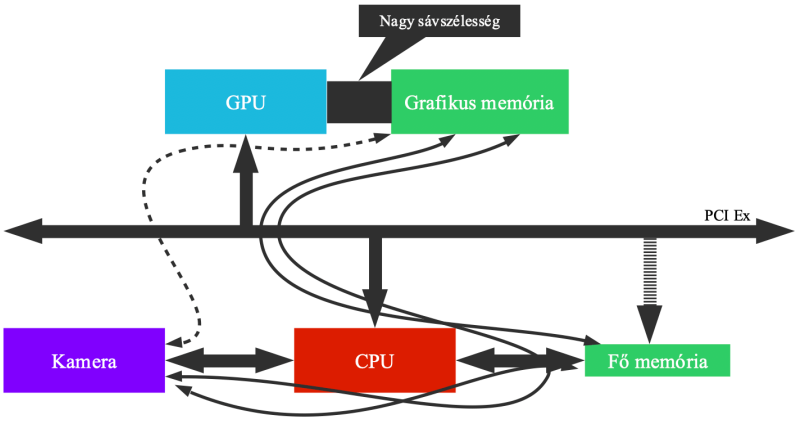
- **A feldolgozás szétosztása több processzorra:** ennek a módszernek a lényege, hogy a rendszerben több, független műveletvégző egység áll rendelkezésre, és emiatt lehetőség nyílik arra, hogy akár a feldolgozandó adatot, akár az elvégzendő műveleteket ezek között az egységek között szétosszuk.
- **Adatfolyam alapú feldolgozás:** ebben az esetben szintén nagy számú műveletvégző egység áll rendelkezésünkre, azonban ezek nem függetlenek, így a feladatokat nem tudjuk elosztani ezek között, az adatmennyiséget viszont igen.
- **Csővezeték technika:** a csővezeték technikát alkalmazó hardverek az egyes műveleteket részegységekre osztják, és azokat a sorban érkező adatokon párhuzamosan hajtják végre. Itt az egy egységgel előbb érkezett adat a feldolgozási lépésekben is eggyel előrébb jár. A csővezeték technika fontos tulajdonsága, hogy egyetlen adat feldolgozását nem gyorsítja meg a teljes feldolgozás átviteli sebességét viszont akár egy nagyságrenddel is növelheti.

16.2. Eszközök

Fontos megemlíteni, hogy a legtöbb speciális hardver eszköz a fent felsorolt paradigmák közül nem kizárólag egyet, hanem általában mindet megvalósítja, csak éppen eltérő mértékben. A többmagos processzorok, számítógép hálózatok és szuperszámítógépek általában az első paradigmát valósítják meg. Tipikusan a Digitális Jelfeldolgozó Processzorok (DSP), a Celluláris Neurális Hálók (CNN), a grafikus processzorok (GPU), és a tenzor processzorok (TPU) tartoznak az adatfolyam feldolgozás területébe. A különböző programozható hardverek (FPGA) és alkalmazásspecifikus hardverek (ASIC) pedig a csővezetéktechnika által elért gyorsításra helyezik a hangsúlyt.

A jelen előadás fókuszában a grafikus feldolgozó egységek (GPU-k) lesznek. Ezek tipikusan az általános célú processzoroknál kevésbé rugalmas eszközök, ugyanis számos feladat hardveresen huzalozva került bennük megoldásra, ezért a teljesítményük akár több nagyságrenddel is jobb lehet. A feldolgozás sebességében ezek általában a programozható hardvereket és az ASIC megoldásokat alulmúlják, azonban ezeknek rugalmassága is kisebb.

GPU-kat minden esetben processzoros rendszerekben használunk. Itt a vezérlő program futtatását minden esetben a CPU végzi, ez végzi a GPU vezérlését is. A kettő közti kommunikációt általában PCI-Express busz segítségével oldják meg, mivel ez egy meglehetősen nagy sávszélességű kommunikációs módszer. Ennek ellenére a két feldolgozó egység dedikált memóriaegységei közti adatmozgatás szokott lenni a grafikus feldolgozás egyik legfontosabb szűk keresztmetszete.



16.1. ábra. *Egy GPU-t tartalmazó rendszer vázlata.*

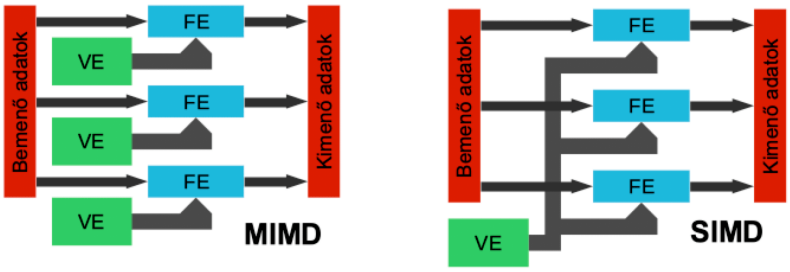
Nem egyértelmű talán első ránézésre, hogy miért is célszerű GPU-k alkalmazása a számítógépes látás feladataira, ugyanis ezek az eszközök elsősorban grafikai feladatok elvégzésére lettek kialakítva. A számítógépes látás ugyan koncepcionálisan a grafika inverzének tekinthető, a két terület meglehetősen hasonlít az elvégzendő műveletek és a feldolgozandó adatok struktúráját illetően. Éppen ezért a GPU-k rendkívül jó támogatást nyújtanak a képfeldolgozás és a számítógépes látás műveleteihez is.

16.2.1. Adatfolyam feldolgozás

A processzorok segítségével történő adatfeldolgozásnak három alapvető architektúrája létezik:

- **Single Instruction Single Data (SISD):** ez a feldolgozás lehető legegyszerűbb változata, ahol egyetlen feldolgozó egység végzi el az összes adat feldolgozását sorosan.
- **Multiple Instruction Multiple Data (MIMD):** ebben a megoldásban az adatokat több független vezérlőegységgel rendelkező feldolgozó egység végzi el, tipikusan a többmagos processzorok alkalmazzák ezt a megoldást.
- **Single Instruction Multiple Data (SIMD):** ebben a megoldásban szintén több feldolgozó egységünk van, azonban ezek nem függetlenek, hanem egyetlen közös vezérlő egységre csatlakoznak. Ennek következtében az egységek ugyanazokat műveleteket végzik el, csak éppen eltérő adatokon. Az architektúra nagy előnye, hogy megegyező költség mellett lényegesen több műveletvégző egység elhelyezése megvalósítható. Ezen felül a SIMD architektúrák programozási paradigmája lényegesen egyszerűbb és kezelhetőbb, mint a MIMD esetben.

A jelen előadásban tárgyalt GPU-k és a később tárgyalandó TPU-k tipikusan a SIMD architektúrát alkalmazzák. A GPU-k esetében az adatfolyam feldolgozást fejlett memóriakezelés, nagy sávszélességű (4096 bit!) memóriák, és csővezeték feldolgozás is segíti. A vezérlő egységek tipikusan kevésbé fejlettek, kevesebb ”spekulációt” végeznek (feltételek jóslása stb.) Fontos eleme a GPU-knak, hogy a globális kártyamemórián kívül az egyes feldolgozó egységek saját regiszterekkel és fejlett cache rendszerrel is rendelkeznek. Érdemes még megjegyezni, hogy a modern processzorok szintén alkalmazzák ezt az elvet: a SWAR (SIMD Within A Register) megoldások lehetővé teszik, hogy egy 64 bites ALU-val rendelkező egység egyszerre 8 db char változóval végezzen műveletet például.



16.2. ábra. *A MIMD (bal) és a SIMD (jobb) architektúrák.*

A GPU-k támogatják továbbá a megegyező architektúrájú kártyák közötti multiprocesszor-jellegű feladat szétosztást is az NVLink technológia segítségével.

Az adatfolyam feldolgozás módszere tehát alapvetően akkor használható jól, amennyiben nagy mennyiségű adaton kell ugyanazt a műveletet elvégezni. Nem érdemes véletlenszerű feldolgozásokra, például szerverek, vagy adatbázisok kiszolgálására alkalmazni őket. Tipikusan könnyen megvalósítható művelet a Map (letérképezés), amikor egy tömb elemein kell függetlenül ugyanazt a műveletet végrehajtani. A redukció művelet során a Map függvényleképezés után az eredményeket egyetlen számba redukáljuk (általában összegzés segítségével). Hasonlóan gyakori a különböző adatszűrések, rendezés és keresés megvalósítása. Érdekes példák még a Gather-Scatter típusú műveletek. Ebben a két esetben úgy olvasunk(Gather)/írunk(Scatter) egy tömbbe, hogy egy másik, kisebb tömbből olvassuk ki az olvasás/írás indexeit.

16.3. GPU Architektúra

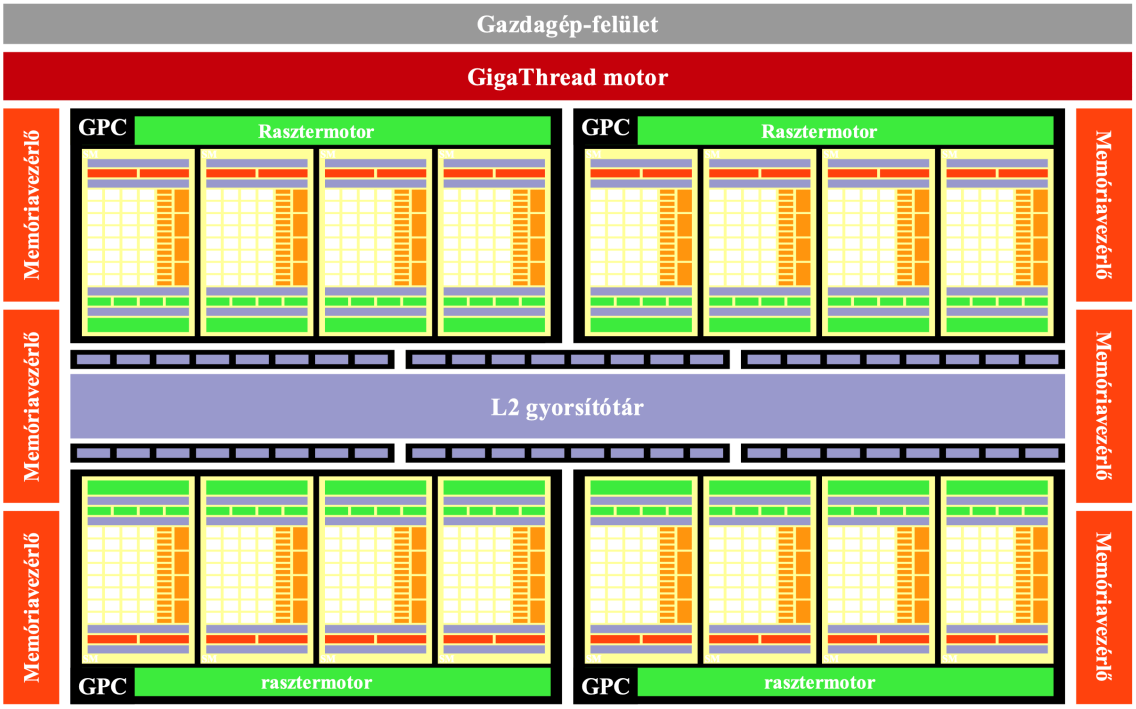
A GPU-k általános felépítése egy közös PCI interfészből és ütemező egységből áll. A globális memóriát több független memóriavezérlő is elérheti, ezek azonban egyetlen közös L2 cache rendszerrel dolgoznak. A GPU feldolgozóegységei klaszterekbe (GPC - Graphics Processing Cluster) rendeződnek, melyeknek saját raszterizálója van, aminek elsősorban grafikus alkalmazásoknál van szerepe. A GPC-k általában 4-8 úgynevezett Streaming Multiprocessor-ból (SM) állnak, amelyben már konkrét feldolgozóegységek találhatók

16.3.1. Streaming Multiprocesszor

A Streaming Multiprocesszor a GPU legfontosabb részegysége, az ebbe tartozó magoknak ugyanis egyetlen vezérlő egysége van. Architektúrától függően egy SM-be 32/64 mag tartozik, melyeknek saját regiszterei vannak. Az SM része még továbbá néhány Special Function Unit (SFU), melyek különböző matematikai műveleteket (szögfüggvények, log, exp) képesek hardveresen megvalósítani. Az SM-nek külön dedikált memóriaolvasó/író úgynevezett Load/Store egységei is vannak, amelyek képesek a globális memóriából a szükséges adatokat a feldolgozástól függetlenül beolvasni. Ezekre azért lehet szükség, mert a globális memória válaszideje akár több száz gépi ciklus is lehet a rengeteg párhuzamos elérés következtében.

Fontos eleme még a Streaming Multiprocesszornak az úgynevezett osztott memória, amely egy olyan memória egység, amelyet az SM összes magja képes elérni (a regiszterek az egyes magokhoz tartoznak). Ezen felül az SM tartalmaz még különböző L1 gyorsítótárakat, melyekből létezik külön az utasításokra és textúrákra szakosodott. Tartalmaz ezen felül külön textúra feldolgozó egységeket és grafikus céláramköröket, melyeknek elsősorban grafikai alkalmazások során van szerepe.

A GPU-k és az SM felépítése változott valamelyest az évek során. Maga a GPU leginkább csak a belepakolt SM-ek számában mutat érdemi eltérést, a Streaming Multiprocesszorok belső felépítése azonban néhány lényeges változáson ment keresztül. Ezek közül a legfontosabb, hogy a GPU-k ma már külön magtípusokat tartalmaznak az egyes számtípusok számára. A legfőbb feldolgozó egység 32 bites lebegőpontos számokon működik, azonban ugyanezek a magok a korábban ismertetett



16.3. ábra. *Egy GPU általános felépítése.*

SWAR elven képesek 16 bites half-precision számok feldolgozására is, amely mély neurális hálók esetén különösen hasznos, itt ugyanis a pontosság kevesebbet számít. Bizonyos architektúrák tartalmaznak 64 bites double egységeket, valamint egész (és fixpontos) aritmetikához használatos magokat is.

16.3.2. Tensor Core

Az egyik legfontosabb újítás ezekben az egységekben az úgynevezett TensorCore, amelyet a Volta és Turing architektúrájú GPU-kban találhatunk meg. Ezek az egységek mátrixszorzásra dedikált áramkörök, minden egyes tenzor mag a következő művelet hardveres megvalósítására képes:

$$C = C + A * B \tag{16.1}$$

Ahol az összes operandus egy 4x4 méretű lebegőpontos mátrix. A tenzor mag nagy előnye, hogy a GPU alapvetően vektor műveletekre lett kifejlesztve, így a mátrixokon végzett operációkat szoftveresen kellett összerakni. A tenzor magok segítségével azonban a mátrixszorzás (és következtében a különböző mély neurális hálók futtatása és tanítása) sebessége egy nagyságrenddel növekedett.

16.4. Programozási modell

Az előző előadás során tárgyaltuk a GPU-k fizikai felépítését, valamint a futási modelljüket, azonban a programozásuk módjáról nem beszéltünk. Éppen ezért a jelenlegi előadás témája a GPU programozásának lehetősége lesz, ezek közül is a CUDA nyelvre fókuszálva.

A GPU programozási lehetőségeit két kategóriába oszthatjuk: ezek közül az első a különböző grafikus programozási nyelvek, a másik pedig az általános célú GPU (GPGPU) programozási nyelvek. Előbbiek közül a két legfontosabb megoldás az OpenGL és a Direct3D. Ezek elsősorban grafikai alkalmazásokra kifejlesztett megoldások, azonban egyszerűbb képfeldolgozási feladatokra (szűrések,

16.4. ábra. *A 2010-es Fermi (bal) és a 2017-es Volta (jobb) architektúrák SM-jei.*

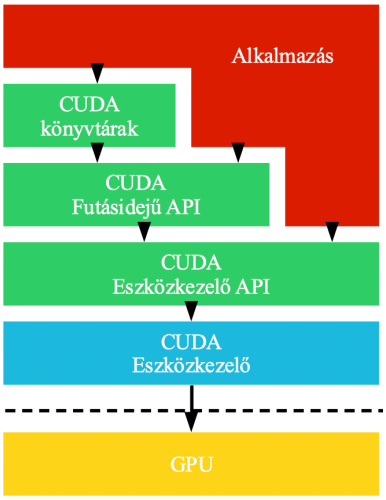
küszöbözés, színtér konverzió) rendkívül jól használhatók. Az OpenGL a legtöbb operációs rendszeren elérhető könyvtárként, és a GLSL nyelvet használja, amely a C++ kiegészítése. Hasonló ehhez a Direct3D által használt HLSL nyelv, azonban ez csak Windows operációs rendszer alatt használható, mivel a Microsoft fejleszti.

Jelen fejezetben sokkal érdekesebbek azonban számunkra a GPGPU programozási nyelvek, vagyis a CUDA és az OpenCL. A CUDA fejlesztését az NVIDIA végzi, ebből kifolyólag csak az ő kártyáikon érhető el. A CUDA egy homogén feldolgozó egységekre alapozó hardvert feltételez, ahol minden eszköz ugyanazt a műveletet végzi el. A CUDA nyelv alapvetően a C/C++ nyelvekre épül, ehhez ad hozzá néhány kiegészítést. Egy CUDA programban az egy GPU mag által futtatott programrészletet (kernelt) kell megírni, ez kerül végrehajtásra minden magon. A CUDA-nak saját fordítója van, amely nvcc névre hallgat.

Az OpenCL (Open Computing Language) ezzel szemben egy nyílt forráskódú, minden GPU-n működő nyelv. Az OpenCL elméletben heterogén eszközökön is működhet, a gyakorlatban azonban ezt nem szokták használni. A CUDA-hoz hasonlóan a C/C++ nyelv kiegészítéseként készül és saját fordítóval rendelkezik, ez azonban GPU gyártónként különböző. A program elkészítésének filozófiája is megegyezik a CUDA-val. A másik lényeges különbség, hogy az OpenCL programok futásidőben is fordulhatnak.

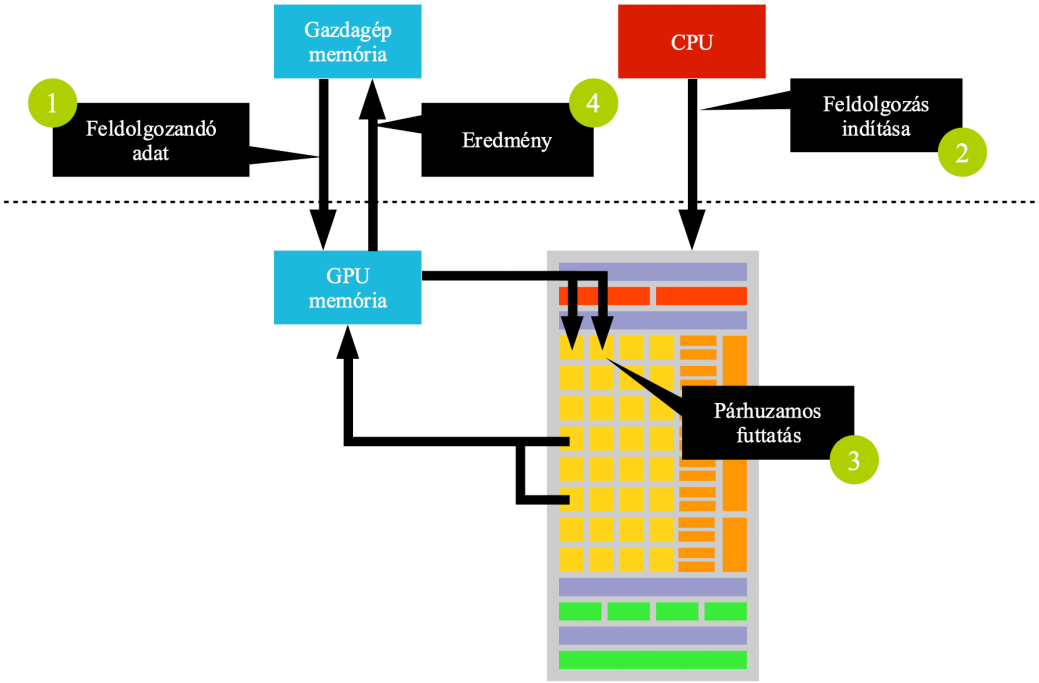
A CUDA nyelv egy több szintből álló architektúra. Ennek legalacsonyabb szintjén a CUDA eszközkezelő (driver) áll, amelyik a GPU-n történő számítás vezérléséért felel. Ehhez a driverhez létezik programozási felület (API), azonban ez egy rendkívül alacsony szintű hozzáférési lehetőség, amelyet csak a legritkább esetben használunk. Leggyakrabban a CUDA futásidejű API-ját használjuk, valamint az arra épülő CUDA könyvtárakat. Fontos megjegyezni, hogy a driver és a futásidejű API használata kölcsönösen kizárják egymást.

Ahhoz, hogy elkezdhessünk a GPU-n programozni, vizsgáljuk meg először, hogy az általunk írt program hogyan fut a GPU-n. Mint már említettük, a feldolgozást a CPU vezérli, és a feldolgozandó



16.5. ábra. *A CUDA szintjei.*

adat is a gazdagép memóriájában található. Első lépésben az adatot át kell másolni a GPU saját memóriájába, majd a feldolgozást kell elindítani. Mivel a két egység közti adatmozgatás és kommunikáció költséges, ezért ezeket célszerű minimalizálni a sebességnövelés végett. Ezt követően a GPU elvégzi a párhuzamos program futtatását, és a végeredményt visszamásolhatjuk a gazdagép memóriájába.



16.6. ábra. *A GPU program futásának módja.*

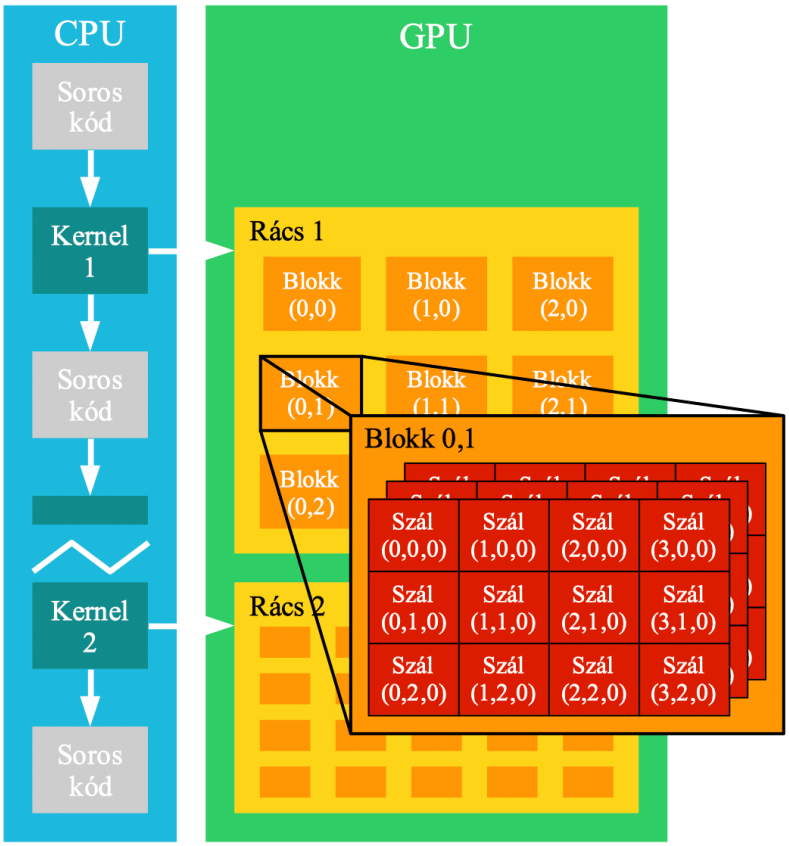
16.4.1. Futási modell

A GPU-n futó programot kernelnek hívjuk, melyből a legtöbb architektúra esetében egyszerre egyetlen futhat. A kernel program alapegysége a szál, amely az egyetlen magon és egyetlen adategységen futó program. A szálakat blokkokba rendezzük, míg a blokkokat pedig egy rácsba (grid). A kétszintű rendezés oka az, hogy az ugyanabban a blokkban található szálak garantáltan ugyan-

azon az SM-en kerülnek végrehajtásra, addig a rácsban található blokkok tetszőlegesen kerülnek szétosztásra az SM-ek közt. Ez különösen fontos, ugyanis ennek következtében az egy blokkban található szálak ugyanazt a közös memóriát látják (hiszen ez fizikailag az SM-ben található), míg a különböző blokkok eltérő közös memóriát látnak.

További különbség ezen felül, hogy a blokkok mérete véges: maximum 1024 szálat tartalmazhatnak, míg a rácsban akármennyi blokk lehet. Ezen felül a blokkban a szálakat rendezhetjük egy, két és három dimenzióba, addig a rács maximum két dimenziós lehet. Ennek fizikai szempontból nincs jelentősége, a többdimenziós rendezés azonban nagymértékben megkönnyíti majd a programozás feladatát.

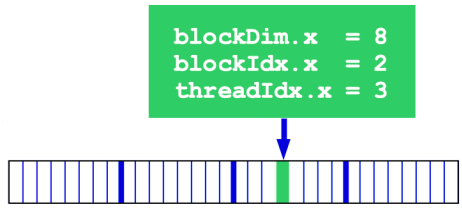
A futási modell utolsó fontos egysége a warp (fonat), amely az adott SM-en egyszerre futó 32/64 szálat jelképezi. Természetesen egy blokk több (maximum 32) fonatból állhat, de egyszerre csak 32/64 szál tud futni. Ez az egység azért fontos, ugyanis ha egy fonaton belül divergens kód adódik (vagyis egyes szálaknak más utasítást kellene végrehajtani), akkor a többi szál addig kénytelen várni, amíg ez a kódrészlet lefut. Éppen ezért a divergens kódot (pl. if-else) a fonaton belül érdemes kerülni.



16.7. ábra. *A CUDA futási modellje.*

Rendkívül fontos speciális változók a kernel index változók, melyek az általunk írt kernelből elérhetők. Ezek az értékek megadják, hogy az adott szál éppen hányadik blokk hányadik szála. Ezek segítségével lehet általános képlet útján megadni, hogy az adott szál a feldolgozandó adattömb hányadik elemét dolgozza fel. Ehhez rendelkezésre állnak a rács és a blokk méretét jelző változók is.

```
1 dim3 gridDim; // rács dimenziója blokkokban (max 2D)
2 dim3 blockDim; // blokk dimenziója szálakban (max 3D)
3 dim3 blockIdx; // 32 blokk azonosítója a griden belül
4 dim3 threadIdx; // 64 szál azonosítója a blokkon belül
5 dim3 warpSize; // 32 fonat mérete (jelenleg mindig 32)
```

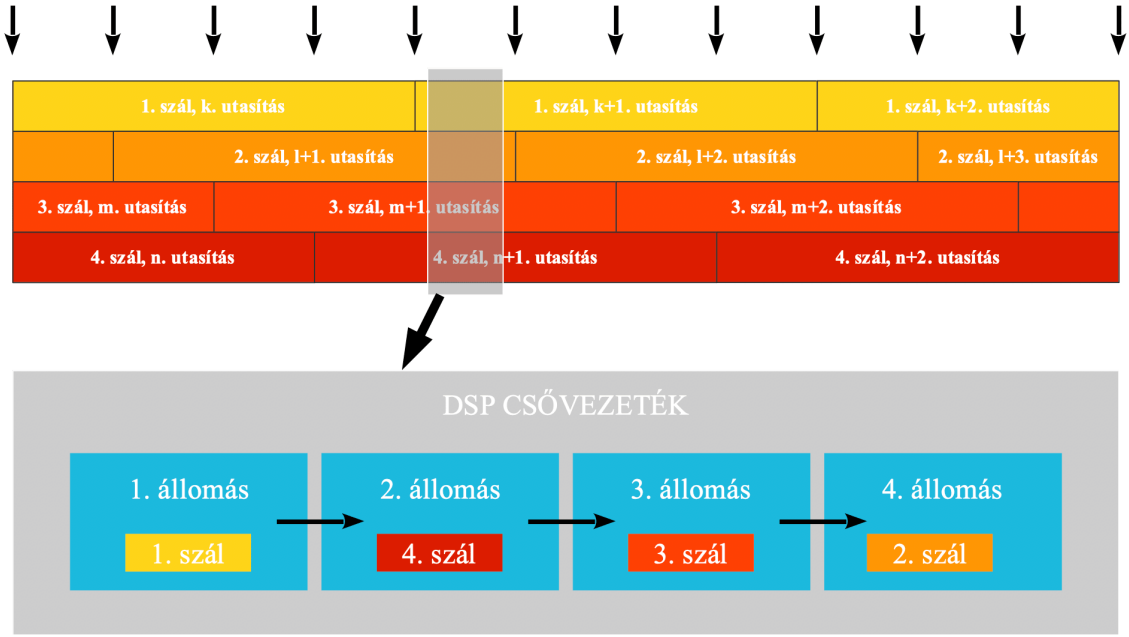


16.8. ábra. *A kernel index számítása.*

Egy tipikus példa ezen változók használatára az alábbi függvény, amely egy tömb minden elemét egy adott konstans értékre állítja.

```
__global__  void assign( int* d_a, int value )
{
     int idx = blockDim.x * blockIdx.x + threadIdx.x;
     d_a[ idx ] = value;
}
```

Érdemes megjegyezni, hogy az SM-nek számos regisztere van, így a fonatok közti taszkváltást nulla többlet teher mellett képes elvégezni, sőt általában egy SM több blokkot is tud könnyedén futtatni. Fontos azt is tudni, hogy az egyes CUDA magok a DSP egységekhez hasonló csővezetéktechnikával rendelkeznek, vagyis képesek több fonatot is párhuzamosan futtatni így, hogy a csővezeték állomásaiba nem az ugyanannak a programnak az egymás után következő lépéseit, hanem a különböző szálak lépéseit töltik be.



16.9. ábra. *A csővezeték többszálas futásra való használata.*

16.4.2. Kommunikáció

Fontos említést tenni még a GPU-n futó szálak közti kommunikáció megoldásáról is, ez ugyanis elengedhetetlen a hatékony programok írásához. Ezek közül az első a blokkon belüli szinkronizációs gát:

```
__shared__ int scratch[blocksize];
scratch[ threadIdx.x ] = value;
__syncthreads__()
leftValue = scratch[ threadIdx.x - 1 ]
}
```

Ezt a módszert elsősorban akkor szoktuk használni, amikor a blokkon belül minden szál valamilyen a többi szál számára releváns információt ír az osztott memóriába, ami a sorban következő művelethez már szükséges. Ilyenkor létfontosságú elérni, hogy a blokkon belül minden szál elérjen idáig mielőtt bármelyik másik szál továbbhaladhatna.

A korábban említett atomi memóriaműveletek szintén alkalmazhatók szálak közti kommunikáció megvalósítására, hiszen ezek garantáltan végrehajtódnak. Tipikus példa erre például a skaláris szorzás implementációja. Ekkor minden szál kiszámolja az összeg egy tagját, majd ezt atomi memóriaműveletekkel ugyanahhoz az osztott memóriában tárolt számhoz adják hozzá.

Fontos megjegyezni, hogy a CPU oldali program a kernel indítása után aszinkron módon tovább halad, ezért az eredmények felhasználása előtt szükséges lehet a szinkronizálást explicit módon elvégezni az alábbi módon:

```
1 cudaThreadSynchronize()
```

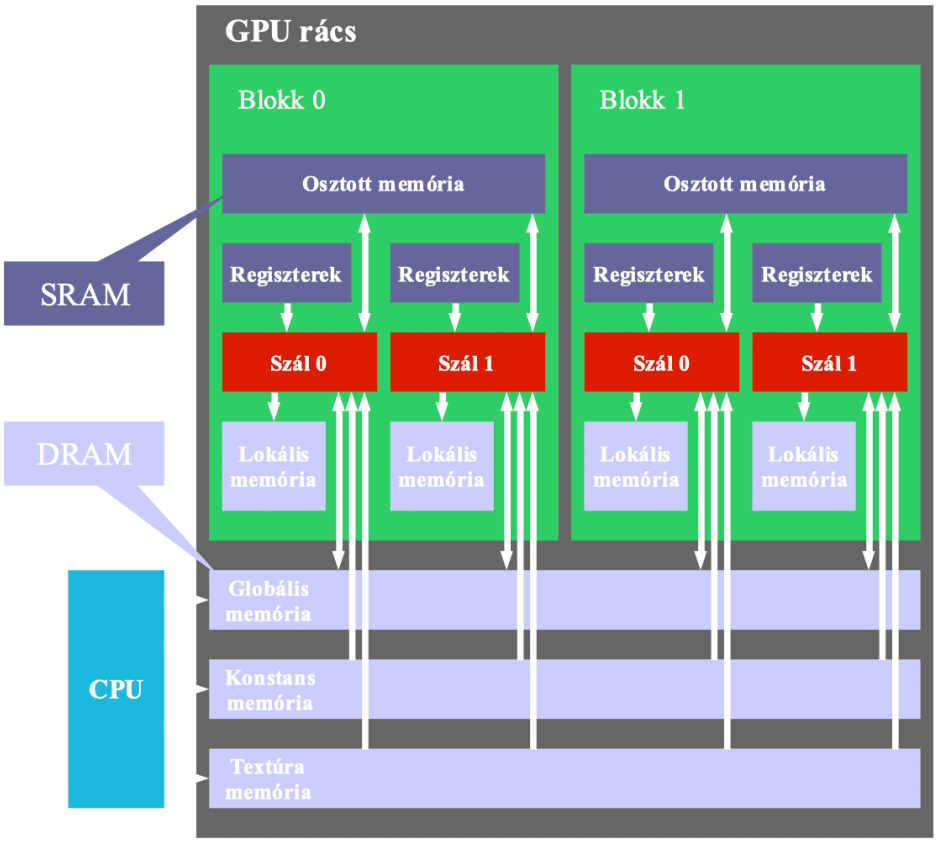
Ez a függvény az éppen futó kernel összes szálának befejezését megvárja. Fontos tudni, hogy újabb kernel indítása előtt ez automatikusan meghívódik, így akkor nem szükséges ezt explicit módon meghívni.

16.4.3. Memóriakezelés

Fontos még tárgyalni a GPU programozás során rendelkezésre álló memóriatípusokat és azok célszerű használatát. A GPU memória ugyanis a feldolgozás egyik legfontosabb szűk keresztmetszetét jelentheti, a számos memória opció pedig első ránézésre lehengerlő lehet. A GPU-k az alábbi memória típusokat különböztetik meg:

- **Regiszterek:** ezek szálanként külön privát memóriaterületek, amelyek fizikailag az SM-ben találhatók, és statikus RAM-ként vannak megvalósítva, vagyis a leggyorsabb memóriatípusról beszélhetünk. Alapesetben a kernelek lokális változói itt kerülnek eltárolásra.
- **Lokális memória:** amennyiben a regiszterek elfogynak a további privát adat itt kerül tárolásra. A lokális memória azonban nevével ellentétben a nem az SM-ben található, ezért fizikailag messze van, ráadásul dinamikus RAM, vagyis meglehetősen lassú típusról van szó.
- **Osztott memória:** az osztott memória szintén statikus RAM, ami az SM-en található, így rendkívül gyors. Ezt a memóriatípust azonban a blokkban lévő összes mag látja, ezért a bonyolultabb elérési és cachelési megoldások miatt valamivel lassabb (főleg írni). Ez a memóriatípus kiválóan használható magok közti kommunikáció megvalósítására, amennyiben elég a blokkon belül kommunikálni.
- **Globális memória:** a globális memória a GPU kártyán külön chipként rendelkezésre álló dinamikus memória. Ez az a memóriatömb, amely a GPU kártyák adatlapján fel van tüntetve, mérete általában 2-16 GB. A legtöbb esetben rendkívül nagy sávszélességű, ennek ellenére az elérése rendkívül lassú tud lenni, amennyiben több ezer szál kíván egyszerre olvasni belőle.
- **Konstans memória:** a konstans memória szintén a dinamikus memóriában helyezkedik el, azonban csak olvasható (szoftveresen). Ennek előnye, hogy a többszálas működés miatt rendkívül bonyolult írás cache kezelés elhagyható, így a működése valamivel gyorsabb, mint a globális memóriáé.
- **Textúra memória:** a textúra memória egy speciális konstans memória, azonban számos hasznos funkciót szolgáltat még számunkra. Ezek közül az egyik, hogy lehetővé teszi számunkra azt, hogy egy-, két- vagy háromdimenziós indexelést használjunk, és az indexek memóriacímmé konvertálását hardveresen végzi. Ezen felül lehetőség van normalizált (vagyis 0 és 1 közötti) indexelés alkalmazására. Ennek külön előnye, hogy kérhetjük a textúra memóriától azt, hogy ha pont két pixel közé indexeltünk, akkor ezeket súlyozva átlagolja össze (konvolúciós szűrés), azonban ez is hardveresen történik.

A textúra memória ráadásul képes 2 vagy 3 dimenziós asszociatív cache megvalósítására is. Ez azt jelenti, hogy ha egy érték bekerül a cache-be, akkor a textúra memória a környezetét is automatikusan beolvassa a cache-be. Ezt azonban nem a memóriacímek alapján, hanem a 2, vagy 3D környezet alapján teszi meg. Ezen felül lehetőség van a textúrából történő kiindexelésre is hiba nélkül. Ekkor több opció közül választhatunk, lehetséges például konstans értéket kérni, vagy akár ismétlődést is.



16.10. ábra. *A CUDA memóriamodellje.*

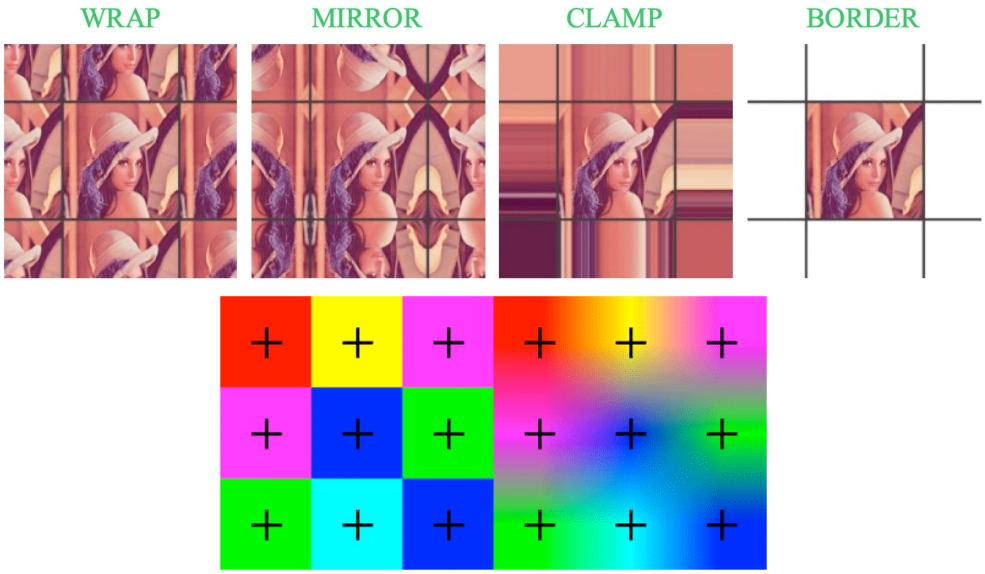
Érdemes még a közös memóriatípusok egyszerre írásáról röviden említést tenni. Az osztott és a globális memória ugyanis alapvetően másféleképpen működnek. Míg a globális memóriában több párhuzamos írás esetében minden írás garantáltan megtörténik (a sorrendjük ugyan nem garantált), addig az osztott memóriában csak egyetlen írás történik meg garantáltan.

16.4.4. Textúrák használata

Külön említést igényel a textúra memória használata, tekintve, hogy számos fontos beállítást/döntést meg kell hoznunk ezek használatához. Fontos tudni, hogy a textúra memóriát CPU oldalról kell definiálni, mivel GPU oldalról ez egy konstans memóriaként viselkedik. A textúra memóriának a következő beállításai lehetnek:

- **Adattípus:** a textúra egyetlen elemének típusa: ezek lehetnek 1-4 dimenziójú vektorok, melyek egész, vagy lebegőpontos számok
- **Dimenzió:** a textúra (és a cachelés) dimenziója (1,2,3)
- **Olvasási mód:** ez a textúra indexelésének típusa: lehet egyszerű tömbindexelés, vagy normalizált, amely esetben a textúra két széle között egy [0,1] tartományba eső lebegőpontos értékkel címzünk.

- **Interpoláció:** beállítható különböző hardveres interpoláció a normalizált indexelés esetén: Ez lehet legközelebbi szomszéd, amely esetben nem történik szűrés. Választhatunk azonban bilineáris interpolációt is.
- **Határkezelés:** amennyiben kiindexelünk a tömbből választhatjuk azt, hogy az olvasás egy előre meghatározott konstanssal, vagy a legközelebbi ténylegesen a tömbbe tartozó értékkel térjen vissza. Választhatjuk azt is azonban, hogy a textúrakezelő vegye úgy, mintha a tömb a határokon túl periodikusan ismétlődne, vagy ugyanezt tükröződő ismétlődéssel.



16.11. ábra. *A különböző határkezelési (felül) és interpolációs (alul) opciók.*

További Olvasnivaló

[45] *CUDA Programming*. Elsevier, 2013. DOI: 10.1016/c2011-0-00029-7. cím: https://doi.org/10.1016%2Fc2011-0-00029-7.

17. fejezet

VPU, TPU, FPGA

17.1. Bevezetés

A korábbi előadások során betekintést nyerhettünk a grafikus feldolgozóegységek működésébe és programozásába. A GPU-k azonban még mindig alapvetően általános célú eszközöknek tekinthetők, amelyekben bizonyos műveletek lettek hardveresen megvalósítva. A jelen előadás során megismerkedünk először a tenzor feldolgozó egységekkel (TPU), melyek kifejezetten mátrixműveletek gyorsítására szolgáló egységek, valamint a programozható hardverek világával.

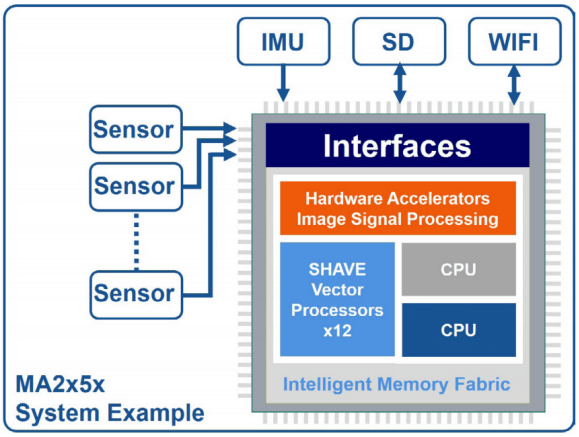
17.2. Vision Processing Unit

Fontos említést tenni a számítógépes látás esetében gyakorta használt Vision Processing Unit (VPU) eszközökről. Ezek általában alacsony fogyasztású néhány RISC architektúrájú maggal, valamint ezen felül számos további speciális párhuzamos feldolgozó egységgel rendelkező eszközök. Ezek a párhuzamos feldolgozók általában alacsony pontosságú lebegőpontos (16/32 bit), vagy fixpontos (8/16/32 bit) műveletek elvégzésére képesek, melyek működését előre huzalozott műveletvégzők is segítik. Ezek a feldolgozók tipikusan VLIW (Very Large Instruction Word) utasításkészlettel rendelkeznek, ami elősegíti a műveletek közti párhuzamosítást.

Ezen felül a VPU-k rendelkeznek lokális, a chipen elhelyezkedő memóriaterülettel is, ami a számítások részeredményeinek hatékonyabb kezelését teszi lehetővé. Végezetül a VPU-k általában számos, digitális kamerák által gyakran használt interfészt is támogatnak. A fentiek alapján a VPU-k felépítése meglehetősen hasonlít az általánosabb célú DSP-re, ezek azonban tipikusan nagyobb pontosságú számítások elvégzésére alkalmasak.

17.3. Tensor Processing Unit

Fontos megérteni, hogy a GPU-k alapvetően vektoros műveletek elvégzésére lettek kifejlesztve (ez alól kivételt képez a tenzor mag). Ezeknek alapvető tulajdonsága, hogy a bemeneteket egyszer használjuk fel egyetlen kimenet íráshoz. Mátrixműveletek esetén azonban az összes bemenetet több művelethez is fel kell használnunk, ráadásul a legtöbb kimeneti változónkat minden műveletvégzés esetében akkumulálnunk kell. Ehhez a GPU programozás során osztott memórián definiált atomi műveleteket kell alkalmaznunk, amik a kernel működését lassítanák. Mivel a mély neurális hálók tanítása és futtatása számos nagy méretű mátrixszorzást igényli, ezért célszerű lenne erre szakosodott hardvert készíteni.



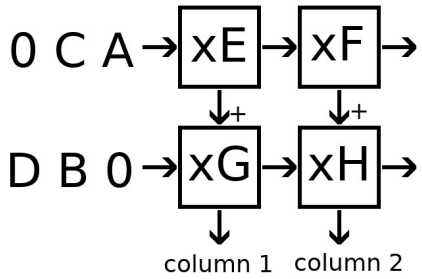
17.1. ábra. *Egy tipikus VPU felépítése.*

17.3.1. Systolic Array

Ehhez első körben vizsgáljuk meg a mátrixszorzást a legegyszerűbb, 2x2-es esetben. A képlet az alábbi:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} * \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{pmatrix} \tag{17.1}$$

Ez szemléletesen megvalósítható négy szorzó és két összeadó egység segítségével az alábbi módon:



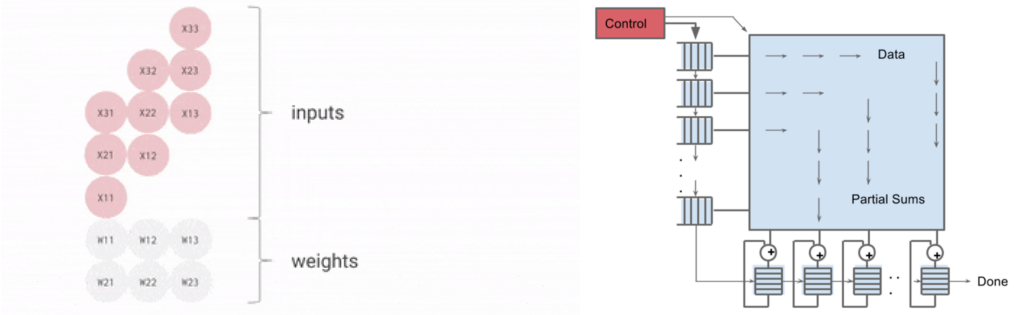
17.2. ábra. *A mátrixszorzás elvégzése.*

Itt a szorzó egységek fixen egy számmal szorozzák a bemenetükre sorban érkező adatokat. A futás végére a szorzat mátrix elemei a kimeneteken oszloponként állnak elő. Itt érdemes megjegyezni, hogy ez a módszer természetesen a kimenetén néhány részeredményt is kiad, összesen azonban 4 ciklus alatt a mátrix összes eleme előáll.

Ezt a végrehajtási architektúrát szisztolikus tömbnek nevezzük. Itt a szisztolikus kifejezés a térbeli párhuzamosítás és a csővezeték-technika együttes intenzív alkalmazását jelenti. Érdemes észrevenni, hogy az alapból köbös műveleti igényű mátrixszorzás ezzel a módszerrel lineáris időben elvégezhető, ami kifejezetten nagy mátrixok esetében jelent hatalmas gyorsulást.

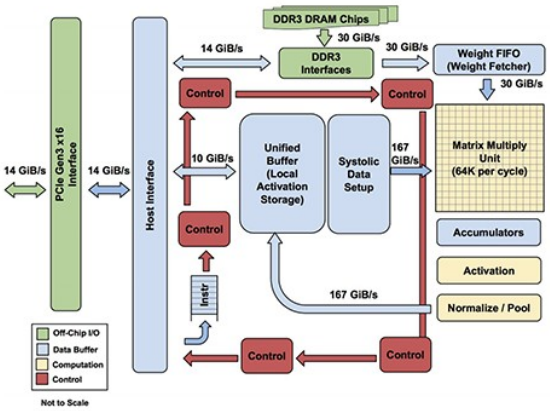
17.3.2. Felépítés

A TPU felépítésének központi eleme a mátrixszorzó egység, amely az imént ismertetett alapelven működik. A mátrixszorzó egységnek külön súly gyorsítótár egysége van, amely csak a neurális háló súlyainak betöltéséért és a mátrixszorzónak való átadásért felelős. A mátrixszorzó egységet a különböző aktivációs függvények huzalozott megvalósítását kínáló aktivációs egységek, és a különböző leskálázások és normalizációk hardveres gyorsítását kínáló norm/pool egységek követik.



17.3. ábra. *A szisztolikus mátrixszorzás (bal) és a TPU mátrixszorzó egységének felépítése (jobb).*

Érdemes még kiemelni, hogy a normalizáló egységek kimenete és a bemeneti adatok/köztes aktivációk tárolásáért felelős egységes buffer között nagy sávszélességű buffer található. Ezt a buffert a mátrixszorzó egységgel egy szisztolikus előkészítő egység köti össze, melynek feladata, hogy az adatot a mátrixszorzáshoz megfelelő formátumba rendezze. Érdemes megjegyezni azt is, hogy míg a bemeneti adatot általában a gazdagép küldi a TPU számára, addig a súlyok lokálisan tárolódnak. Ennek oka, hogy az első generációs TPU alapvetően inferencia (háló futtatás) feladatokra lett kialakítva.



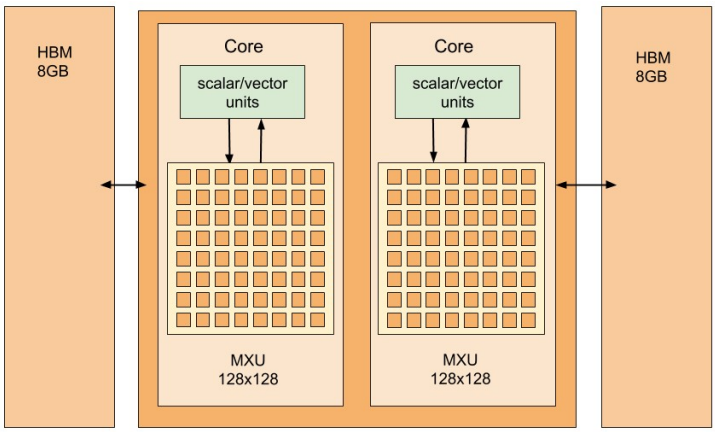
17.4. ábra. *A TPU felépítése.*

A második generációs TPU-k azonban már enyhén módosított architektúrával rendelkeznek. Megfigyelhető, hogy nagy mennyiségű lokális HBM (High Bandwidth Memory) memóriával rendelkeznek, valamint, hogy a korábbról ismert aktivációs, normalizációs és pooling egységek általános skalár/vektor feldolgozó egységekre lettek cserélve. Ez azért történt, mert a második generációs TPU-k már tanításra is alkalmazhatók.

17.4. Programozható hardverek

A GPU-k TPU-k Deep Learning feladatokra rendkívül jól használható egységek, azonban egyéb speciális látási algoritmusok nehezebben implementálhatók velük. Különösképp azok az algoritmusok nem gyorsíthatók ezeken az eszközökön, amelyek nagymértékben hagyatkoznak feltételes végrehajtásokra és elágazásokra. Ezekben az esetekben azonban létrehozhatjuk a saját feladatspecifikus hardverünket valamilyen programozható hardver segítségével.

Programozható hardvereknek számos fajtája létezik, kezdve a kombinációs hálózatok megvalósítására képes PLA (Programmable Logic Array) és PROM (Programmable Read-Only Memory) áramkörökkel. Léteznek sorrendi hálózatok megvalósítására képes eszközök is, ilyen például a



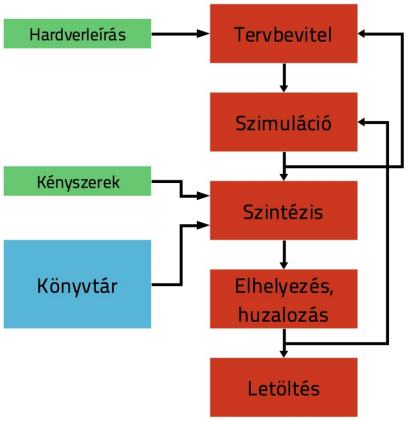
17.5. ábra. *A második generációs TPU felépítése.*

CPLD (Complex Programmable Logic Device). Bonyolultabb hardverek, algoritmusok megvalósítására azonban leggyakrabban FPGA (Field Prgrammable Gate Array) eszközöket szokás használni.

Az FPGA-k olyan speciális hardvereszközök, melyek segítségével elsősorban a csővezeték-technológia, kisebb részben a párhuzamos feldolgozás segítségével tudunk az algoritmusokon gyorsítani. Érdemes megjegyezni, hogy FPGA-kat gyakorta használunk hardvertervezés során, ugyanis az új IC-k készítése rendkívül költséges folyamat. Az elkészült és működő hardverterv esetében azonban lehetséges (és tömeges gyártás esetén célszerű is) a megtervezett hardvert ASIC (Application Specific IC) formájában is létrehozni.

Az FPGA tervezés során a hardvert kezdetben gyakran blokkvázlat szintjén hozzuk létre, majd ezt egy választott hardverleíró nyelv segítségével tudjuk lekódolni. Az FPGA áramkörökhöz tartozik egy szintézer program is amely a leírt hardvert átkonvertálja az adott céleszköz egy - a hardvernek megfelelő - konfigurációjába. A tervezés során általában lehetőségünk van a leírt hardver szoftveres szimulációjára, ahol a tervezési hibák jó része már mutatkozik. Amennyiben a szimuláció sikeres, a következő lépés a hardver szintézis, melynek során a szintézer az általunk leírt hardvert egy könyvtár segítségével megvalósítható hardverré szintetizálja.

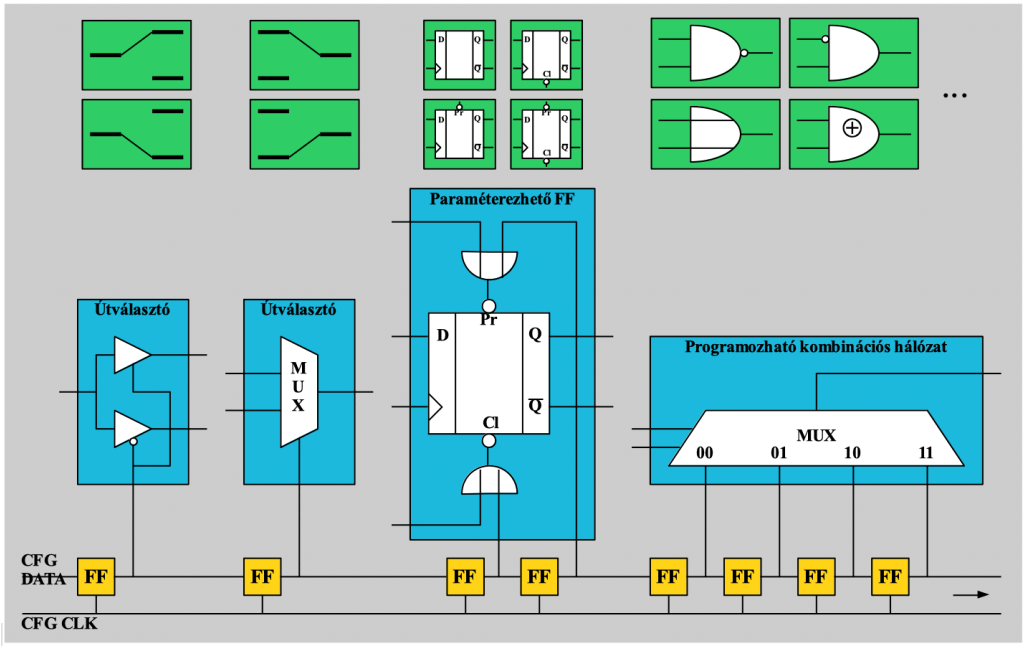
Ezt követően jön az elhelyezés és a huzalozás lépése, amely már a konkrét céleszköz paramétereit is igényli. Ennél a lépésnél kerülnek az egyes egységek az FPGA egyes blokkjaiba elhelyezésre, és itt határozható meg, hogy mi a hardver futásának maximális sebessége (ez az egyes kapcsolódó egységek közötti fizikai távolságtól függ). Az elhelyezés végén előáll a végleges konfiguráció, amely a hardverre már letölthető.



17.6. ábra. *A FPGA tervezés folyamata.*

17.4.1. Architektúra

Az FPGA architektúra tárgyalásának elején célszerű megérteni, hogy miként lehetséges egy hardvert programozhatóvá tenni. Az FPGA minden egyes programozható eleméhez tartozik legalább egy konfigurációs flip-flop, amelyek egy bites értéket képesek tárolni. Ezek sorosan vannak egy adatvezetékre felfűzve, az órajelet pedig egy külön a konfiguráláshoz dedikált generátortól kapják. A konfiguráció felöltésekor a flip-flopok órajelet kapnak, a konfigurációs értékeket pedig sorosan (a flip-flopok sorrendjével megegyező sorrendben) küldjük át az adatvezetéken. Ilyenkor a flip-flopok egyetlen hatalmas shift regiszterként viselkedve minden elemhez eljuttatják a saját konfigurációját. A konfiguráció befejeztével az órajelet kikapcsoljuk, így minden flip-flop konstans memóriaként tartja ezt a konfigurációs értéket, amelyet különböző hardverelemek bemenetként használnak fel. Az alábbiakban látható néhány példa ilyen elemekre:



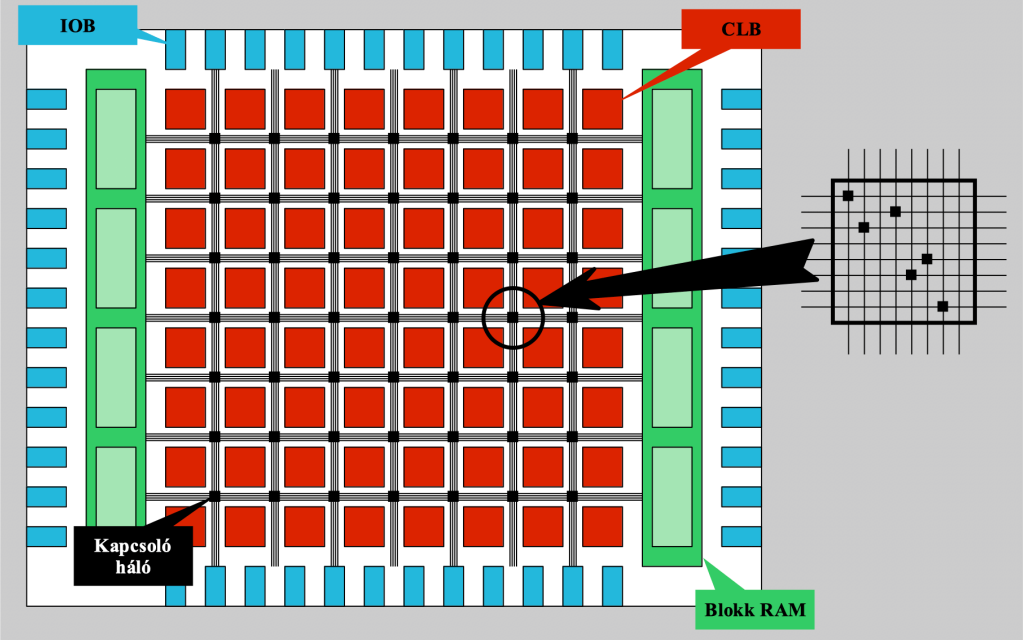
17.7. ábra. *A programozható hardverek megoldása.*

Egy teljes FPGA chip számos ilyen konfigurálható elemet tartalmaz, amelyek úgynevezett konfigurálható logikai blokkokba (CLB - Configurable Logic Block) szerveződnek. A CLB-k között egy programozható sínrendszer van, melynek segítségével az egyes CLB-k tetszőelgesen köthetők össze. Az FPGA-hoz tartoznak még úgynevezett blokk RAM egységek, amelyek on-chip memóriaként szolgálnak, valamint különböző IO blokkok, melyek segítségével az FPGA chip és a külvilág közti kommunikációs interfész valósítható meg.

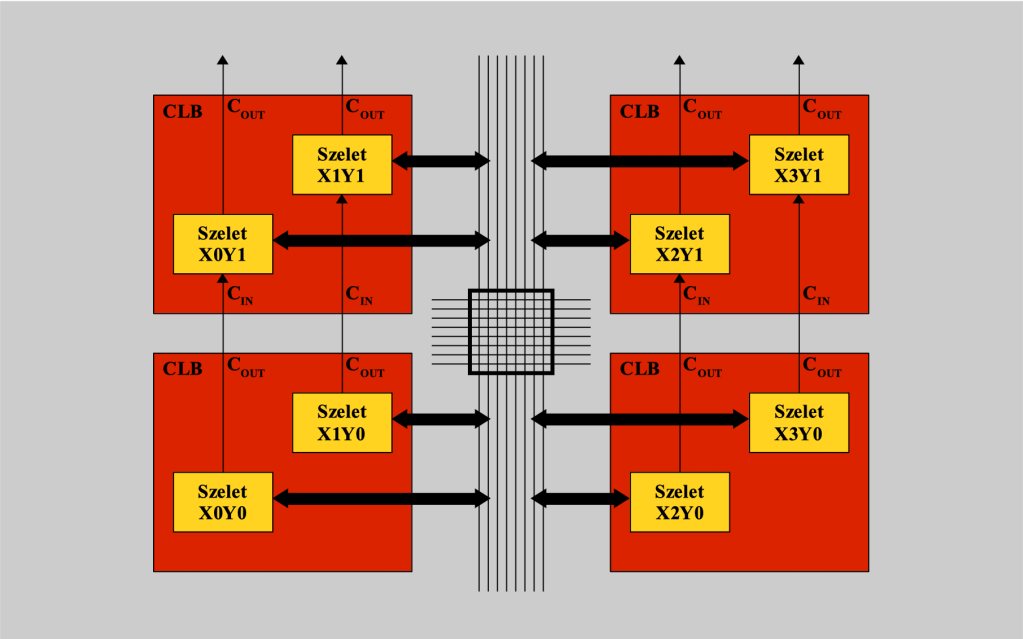
Egy CLB alapvetően két szeletből (Slice) áll, amely a konfigurálható hardverek alapegysége. Ezek a szeletek a szomszédos CLB-k szeleteivel közvetlen (a globális sínrendszertől független) kapcsolatban is állnak, amely ennek a sínrendszernek a terhelését csökkenti nagy mértékben. A CLB-k közti összeköttetések ugyanis az FPGA tervezés egyik szűk keresztmetszete, így a gyakori, és egyszerűen megoldható kapcsolatokat érdemes külön, dedikáltan megvalósítani.

17.4.2. Szeletek

Minden szelet tartalmaz több look-up table áramkört, amelyek 1 bites értékeket tárolnak. Ezen felül minden szelet tartalmaz több, tetszőlegesen konfigurálható flip-flopot is. Ezen flip-flopok kezdeti értéke, szinkron/aszinkron viselkedése, órajel engedélyezése, stb. tetszőlegesen megválasztható. Minden szelethez tartozik továbbá néhány, az aritmetikai műveleteket kiegészítő eszköz: ilyen például az átvitelképző egység, vagy a számláló. Található még ezen felül számos multiplexer



17.8. ábra. *Az FPGA általános felépítése.*



17.9. ábra. *A CLB-k általános felépítése.*

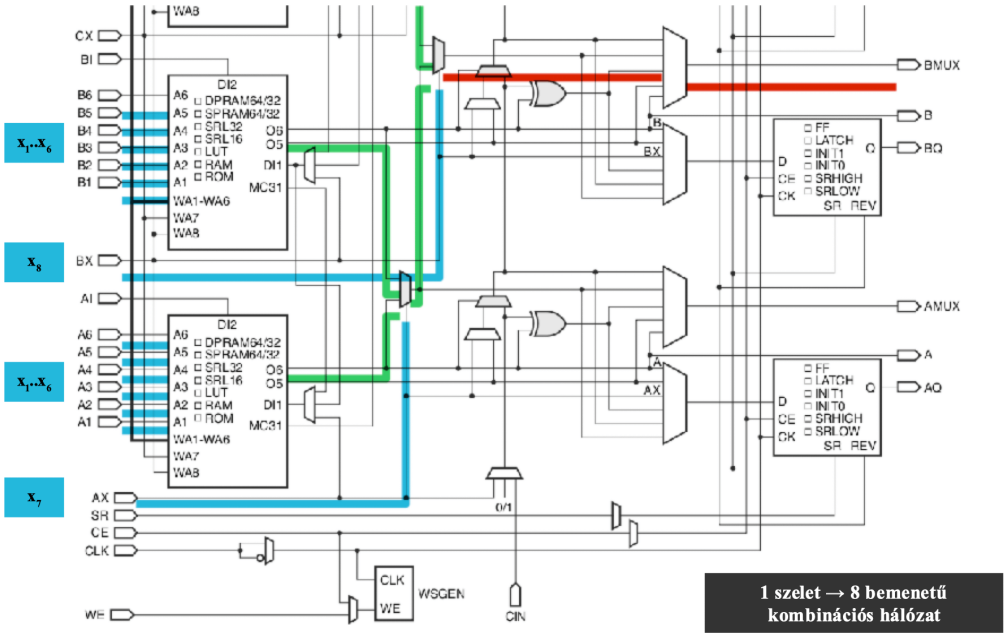
is a szeletekben, mellyel útvonalválasztás, kombinációs függvények kiegészítése, és egyéb speciális feladatok is végezhetők.

Fontos külön említést tenni a szeletekben található LUT egységek működéséről. Ezek az egységek alkotják ugyanis egy programozható szelet magját. LUT segítségével megvalósíthatunk egyszerű logikai függvényeket, melyek bemenetszáma 5-8 között változhat. Erre a bővítésre a slice multiplexerei használhatók. Természetesen egyetlen szeletben véges mennyiségű LUT található, így a több bemenetű logikai hálózatból kevesebb fér bele egy szeletbe. A kívánt működés eléréséhez csupán annyit kell tennünk, hogy a hálózat igazságtábláját beletöltjük a LUT memóriaelemeibe.

Érdemes még megemlíteni, hogy ezen felül a LUT használható RAM, vagy (virtuálisan) ROM elemként is. Ezek 1 bites memóriacellaként fognak viselkedni, azonban az elérésük lényegesen

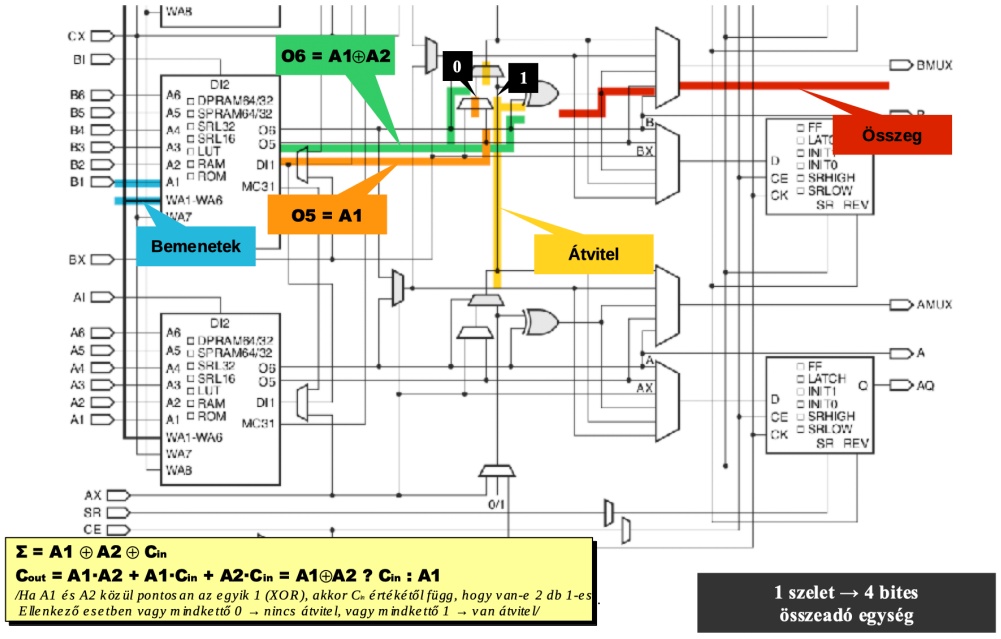
gyorsabb a blokk RAM-hoz képest. Felhasználhatók a LUT egységek még shift regiszterként is, amely hardverek esetén gyakran alkalmazott egység.

Vizsgáljunk meg röviden néhány példát a szeletek használatára. Ezek közül a legegyszerűbb a kombinációs hálózat megvalósítása: itt a 8 bemenet közül hatot a LUT-ok bemenetére kapcsolunk, a másik kettő pedig a LUT-ok közül választó multiplexereket vezérli:



17.10. ábra. *Egy 8 bemenetű kombinációs hálózat megvalósítása (megjegyzés: van még 2 db LUT a képen kívül).*

Ennél valamivel bonyolultabb az összeadó megvalósítása. Itt a LUT-ot a XOR művelet megvalósítására használjuk fel, míg a carry logika megvalósítását a LUT mögött található kiegészítő áramközök segítségével végezzük el az alábbi módon:



17.11. ábra. *Az összeadás megvalósítása.*

17.4.3. Memória Opciók

Érdemes még említést tenni az FPGA-ban általában elérhető memória típusokról. Ezek közül kettőt már röviden tárgyaltunk: a szeletekben lévő elosztott memóriát és a Blokk RAM-ot. Az elosztott memória tipikusan bites szervezésű és kis méretű: szeletenként 64-256 bit között változhat típusonként. Lokális elhelyezkedése miatt rendkívül nagy sebességű, és lehetőséget ad aszinkron módon történő olvasásra is.

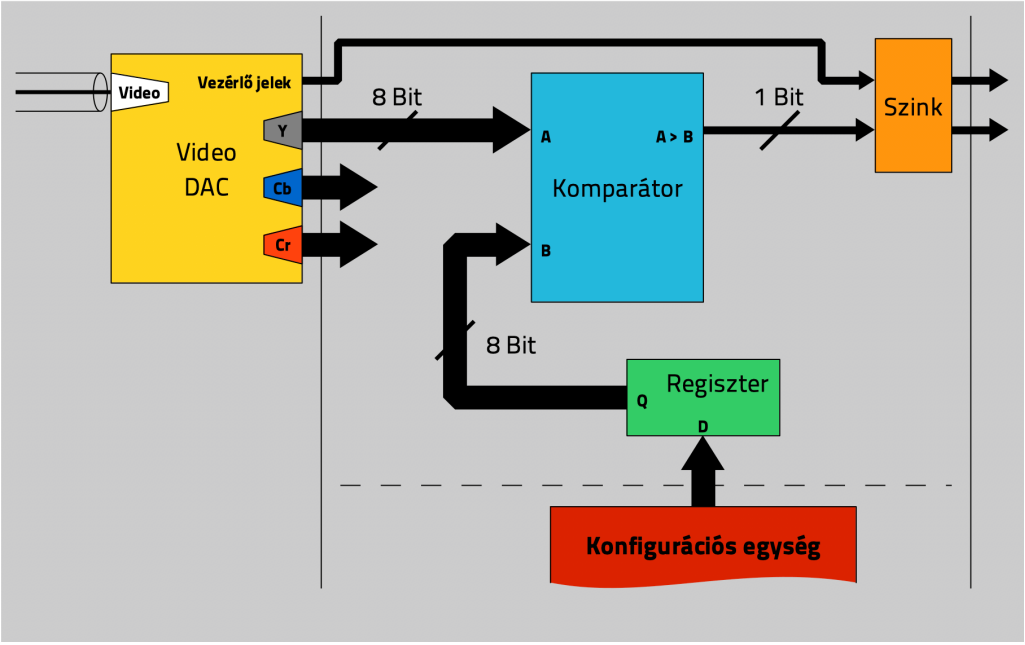
A blokk RAM az FPGA szilícium chipjén található, és közepes méretű (18-36kbit) RAM. Többféle bitszervezést is támogat (1,2,4,8,9,18,36 és 72), melyekbe a paritás bit is beleértendő. Fontos tulajdonsága, hogy az írás és az olvasás is szinkron módon történhet csak. Ez a memóriatípus használható FIFO üzemmódban is. Az FPGA kártyáknak a gyakorlatban szokott lenni egy külső memória chipjük is, ami már egy különálló DDR memória, és közösen használható az FPGA kártyát vezérlő mikroprocesszorral. Ez a memória a három közül a legnagyobb, egyben a leglassabb is.

17.5. Példák

Végezetül nézzünk néhány példát képfeldolgozási műveletek hardveres megvalósítására.

17.5.1. Küszöbözés

A legegyszerűbb feladat a küszöbözés, melynek során a példában azt feltételezzük, hogy a pixelek közvetlenül a kamerából jönnek, sorosan, jelenleg YCbCr színtérben (ez kamerák esetében gyakori lehetőség).

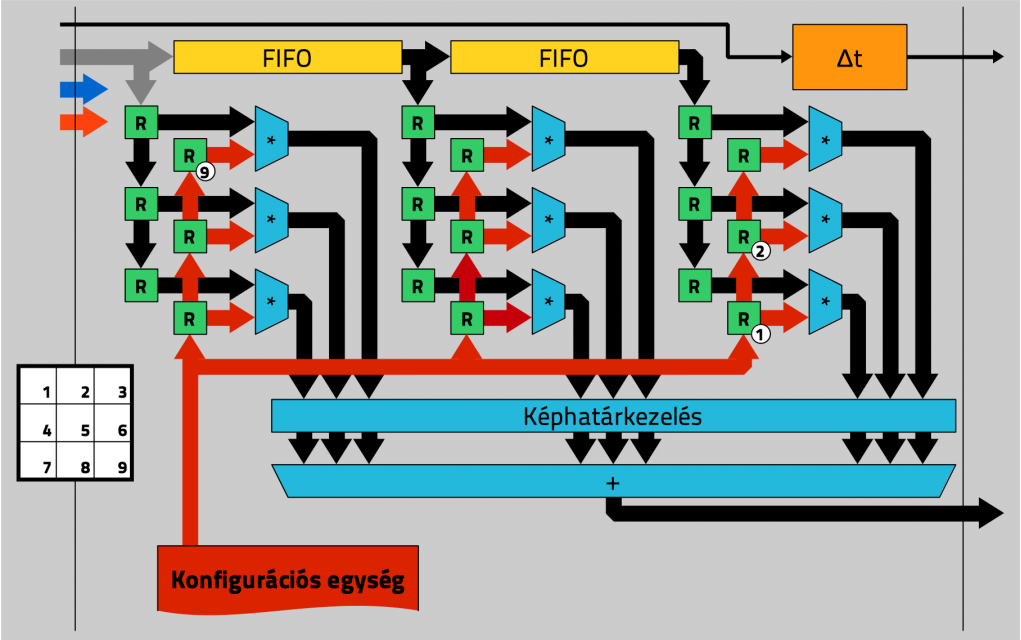


17.12. ábra. *A küszöbözés blokkvázlatos megvalósítása.*

17.5.2. Konvolúció

Konvolúció esetén már nehezebb a dolgunk, hiszen nem csak egyetlen pixelt kell felhasználunk a művelet számításához, hanem az egy sorral előbb és később jövő képpontokat is.

Ebből kifolyólag a hardverünk nagy számú FIFO-t tartalmaz, amelyek a pixeleket 1-1 egész képsornyival késleltetik. A szűrő súlyait a konfigurációs egységből kapják meg a szorzók, a pixelek sorrendjével fordított módon. Fontos megemlíteni, hogy a konvolúció megvalósításánál a képhatárkezelést is meg kell valósítani, vagyis a kép területéről kilógó helyekről érkezett szorzatokat el kell vetni.



17.13. ábra. *A konvolúció blokkvázlatos megvalósítása.*

17.6. Magas szintű tervezés

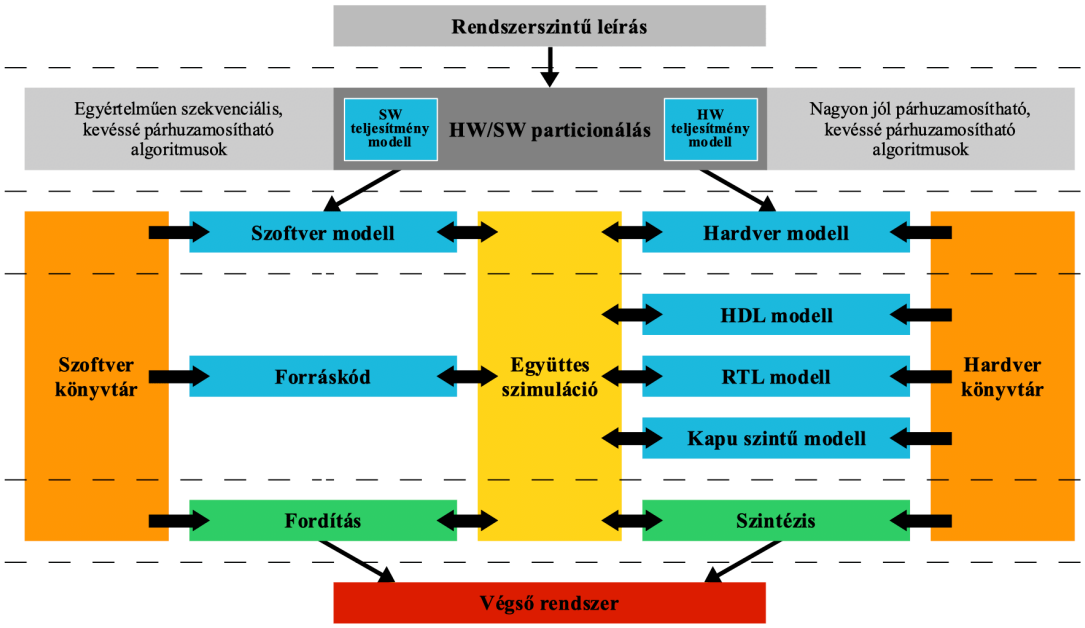
Az előadás utolsó részeként beszéljünk röviden az FPGA segítségével történő hardvertervezés néhány fontos kérdéséről. Fontos, hogy FPGA tervezés során a hardver és a szoftver tervezését együttesen, egymásra épülve érdemes elvégezni. Ehhez az elvégzendő feladatokat először elosztjuk a két eszköz között. Az egyértelműen szekvenciális algoritmusokat a CPU-n, míg a jobban párhuzamosítható és gyorsítható eljárásokat hardverből érdemes megvalósítani. A tervezés minden lépése során célszerű a hardver és szoftver rendszert közösen szimulálni.

17.6.1. Adatutak rendszere

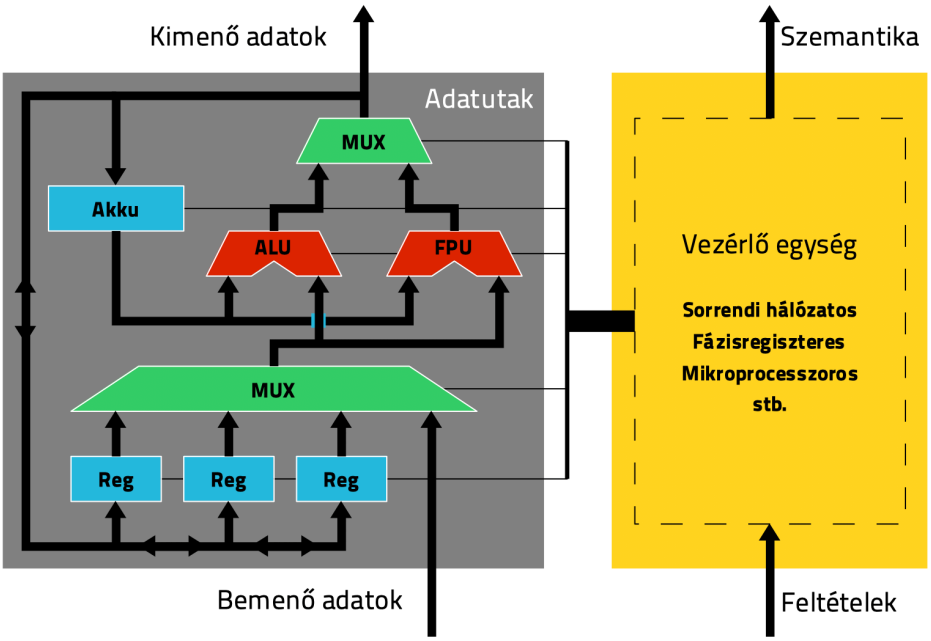
Az FPGA segítségével történő feldolgozás leírható úgynevezett adatút-rendszerként, amelyben az egyes adatelemek különböző funkcionális elemeken áthaladva kerülnek feldolgozásra. A feldolgozás menetét egy külön vezérlőegység felügyeli, amely a számára rendelkezésre álló feltételek alapján az egyes adatelemek konkrét útját változtathatja. Ez a vezérlő egység lehet egy egyszerű sorrendi hálózat, de akár egy komplett mikroprocesszoros rendszer is.

17.6.2. Csővezeték

Az adatutak rendszerének alapvető eleme az algoritmikus csővezeték. Ezen a csővezetéken a feldolgozandó adathalmaz sorosan, elemenként halad át, amely képek esetén azt jelenti, hogy az egyes műveleteket pixelenként végezzük sorosan. Ez azt eredményezi, hogy habár a teljes kép feldolgozása nem feltétlenül gyors (itt párhuzamosításról nem beszélünk), azonban a csővezetéket a pixelek soros továbbítási interfészébe beleépítve a feldolgozás csak minimális (az alábbi példa esetén például néhány sor és pixel nagyságrendű) késleltetést okoz.



17.14. ábra. *A hardver és szoftver együtt tervezésének folyamata.*



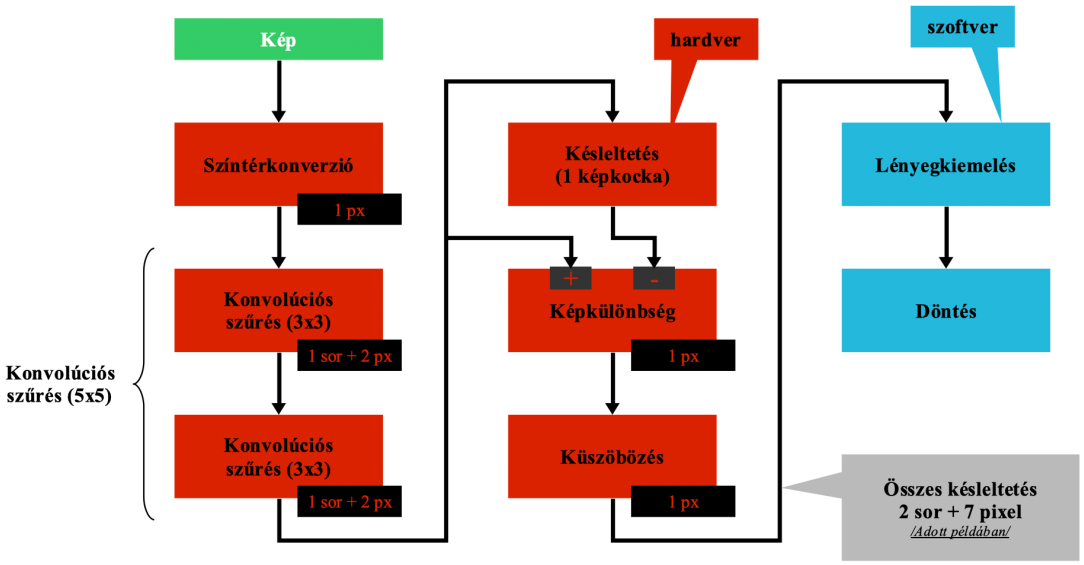
17.15. ábra. *Az adatutak rendszere.*

17.6.3. Újrakonfigurálás

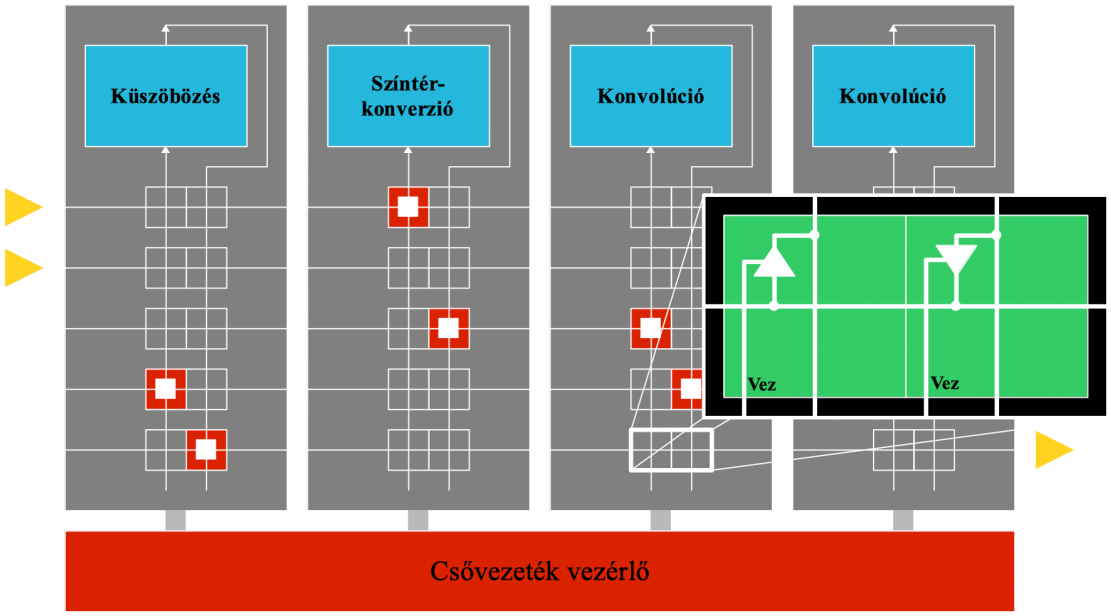
A csővezetékek fontos tulajdonsága lehet, hogy az egyes adatok útja változtatható legyen, vagyis programozható csővezetékünk adódjon. Ehhez az egyes algoritmikus blokkok közé egy konfigurálható buszrendszert kell elhelyezni.

Vezérelhető csővezetékből létezik az egyszerű megoldáson felül azonban úgynevezett szuperskalár csővezeték is. A szuperskalár feldolgozás azt jelenti, hogy a hardver egyszerre több, párhuzamos feldolgozási csővezetéket is használ és menedzsel. Ilyen csővezeték segítségével elvégezhető adat- vagy feladatszintű párhuzamosítás is az FPGA-n.

Létezik ezen felül olyan megoldás is, amelyben nem csak maga a csővezeték, hanem a teljes hardver



17.16. ábra. *Egy küszöbözött képdifferenciát megvalósító hardveres csővezeték.*

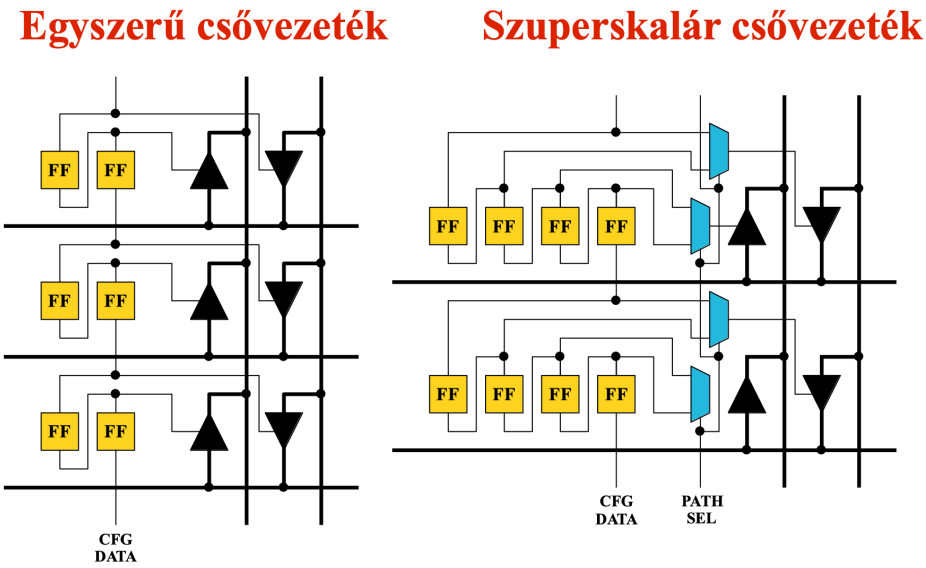


17.17. ábra. *Az újrakonfigurálható csővezeték.*

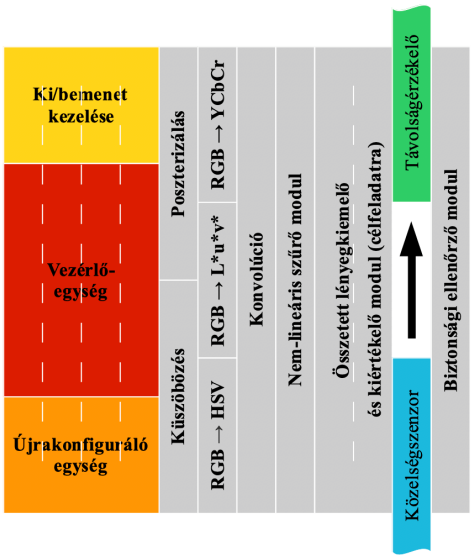
is újrakonfigurálható lesz. Erre elsősorban akkor lehet szükség, ha a feldolgozási feladat bizonyos események hatására változhat, vagy ha több feldolgozó egység között adaptívan szeretnénk a feladatokat kiosztani. Ez utóbbit megtehetjük a terhelés, vagy a fogyasztás optimalizálása miatt, vagy esetleg azért, hogy eltérő prioritású feladatokat más teljesítményű eszközökre is oszthassunk (itt a prioritás változhat menet közben).

További Olvasnivaló

[46] S. Kilts, *Advanced FPGA Design*. John Wiley & Sons, Inc., 2007. jún. DOI: 10.1002/9780470127896. cím: https://doi.org/10.1002%2F9780470127896.



17.18. ábra. *A hagyományos (bal) és a szuperskalár (jobb) csővezeték vezérlők.*



17.19. ábra. *Az újrakonfigurálható hardver.*

Köszönetnyilvánítás

Köszönöm Reizinger Patriknak, hogy elkészítette a jegyzet angol nyelvű változatát.

További köszönet illeti Dr. Vajda Ferencet, aki a tárgy eredeti tematikáját kidolgozta, valamint a jegyzetben szereplő ábrák egy részét elkészítette.