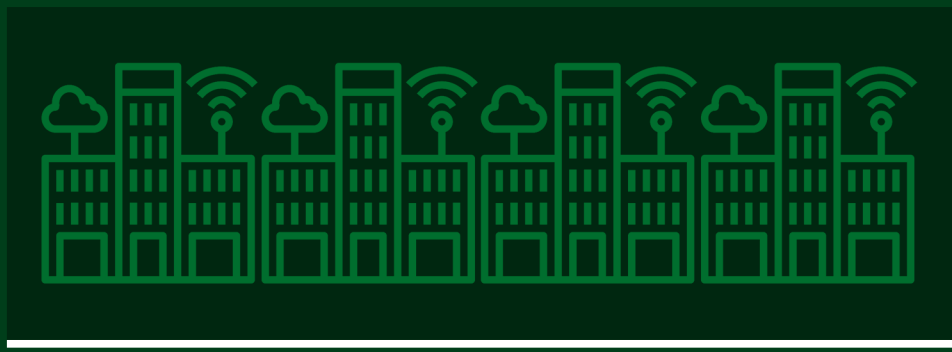
OKOSVÁROS-TECHNOLÓGIÁK IX.

Számítógépes látórendszerek az e-közigazgatásban: gyakran használt algoritmusok és megoldások

Kismonográfia



Szemenyei Márton

Nemzeti Közszolgálati Egyetem
Budapest

SZÉCHENYI 2020
MAGYARORSZÁG KORMÁNYA
Európai Unió
Európai Szociális Alap
BEFEKTETÉS A JÖVŐBE

OKOSVÁROS-TECHNOLÓGIÁK
A technológia fejlődésének irányai és hatása
IX. kötet

Sorozatszerkesztő:
Sallai Gyula

Szemenyei Márton

SZÁMÍTÓGÉPES LÁTÓRENDSZEREK AZ E-KÖZIGAZGATÁSBAN: GYAKRAN HASZNÁLT ALGORITMUSOK ÉS MEGOLDÁSOK

KISMONOGRÁFIA

Nemzeti Közszolgálati Egyetem
Közigazgatási Továbbképzési Intézet
Budapest, 2020

A kiadvány a KÖFOP-2.1.2-VEKOP-15- 2016-00001 „A jó kormányzást megalapozó közszolgálat-fejlesztés” című projekt és a BME–NKE „Okos város – okos közigazgatás” kutatóműhely (2017/162/BME-VIK) keretében készült el és jelent meg.

Szerző:
Szemenyei Márton egyetemi tanársegéd, BME

Lektor:
Dr. Jakab László egyetemi tanár, BME

A kézirat lezárásának dátuma:
2018. szeptember 25.

TARTALOMJEGYZÉK

BEVEZETÉS

A számítógépes látás a számítástudomány egyik legrohamosabban fejlődő területe, amelynek egyre több gyakorlati felhasználása létezik. Ez a tendencia nem meglepő, hiszen az ember az érzékszervei közül a szemére hagyatkozik a leginkább a napi feladatainak ellátásakor. Ebből következik, hogy ha a számítógépes látás algoritmusai képesek megközelíteni, vagy akár felülmúlni az emberi látás képességeit, akkor számos fontos feladatot leszünk képesek automatizálni.

Könnyen belátható azonban az is, hogy a gyakorlatban ez rendkívül nehéz, mivel általában olyan feladatokat tudunk könnyedén algoritmusok formájában leírni, ahol az elvégzés menetét pontosan, tudatos szinten megértjük. A szemből érkező jelek feldolgozásának azonban a jelentős része a tudatalatti szinten történik, így aligha tudjuk ezeket a folyamatokat könnyedén megérteni és algoritmusok formájában lemásolni.

Az előző kötetben megismerkedtünk a számítógépes látás alapjaival, illetve számos egyszerű algoritmussal, amelyek különböző alacsony szintű feladatokat tudtak elvégezni. Fontos azonban hangsúlyozni, hogy ezek az algoritmusok bár rendkívül hasznosak, pontosságuk és teljesítményük nem közelíti meg az emberi látás szintjét. Könnyen beláthatjuk, hogy ha ezt a szintet szeretnénk megközelíteni, vagy akár túllépni, akkor a kamerák jeleit feldolgozó algoritmusoknak valamilyen intelligenciát, tanulóképességet kell adnunk.

A jelenlegi kötet egyik jelentős fókusza a mesterséges intelligencia, illetve ezen belül is a gépi tanulás módszereinek alkalmazása a számítógépes látás területén. A kutatóműhelyek körében a tanuló látás, ezen belül is az úgynevezett mély tanulás (deep learning) módszerei az utolsó tíz év első számú áttörését és fejlődési potenciálját jelentették. Ezek a módszerek az utóbbi néhány évben egyre hangsúlyosabban jelentek meg különféle ipari alkalmazásokban is. Érdemes megjegyezni, hogy a mély tanulás alapú látás néhány alapfeladat terén képes megközelíteni, vagy akár túllépni az emberi látás pontosságát.

A számítógépes látás egyszerűbb algoritmusainak másik alapvető hiányossága, hogy a valódi, háromdimenziós világ egy kisebb dimenziós, torz vetületét használja. Érdemes ugyanis észrevenni, hogy egy kép készítése során az eredeti világ (jelenet) térbeliségére vonatkozó információk egy jelentős része elveszik, vagy éppen torzul. Ebből kifolyólag célszerű lehet bizonyos alkalmazások esetén megkísérelni az eredeti térbeli jelenetet visszaállítani az abból készült képek információja alapján. Ez a megoldás – bár számításigényes – mégis gyakran olcsóbb vagy kivitelezhetőbb, mint egy mélységszenzor használata.

A jelenlegi kötet másik fontos fókusza a térbeli látás algoritmusainak bemutatása és tárgyalása lesz. Ezek a megoldások gyakorta szükségesek, ha az általunk készített látórendszernek valamilyen interakcióba kell lépnie a valós környezettel, mint például a kiterjesztett valóság rendszerek esetén. Alkalmazásuk szükséges lehet olyan esetekben is, amikor közvetlenül van szükségünk a 3D információra, mint például műhold- vagy légi felvételekből történő térképkészítés esetén.

A kötet végén egy külön fejezet keretében fogjuk tárgyalni az ezen és az előző kötetben bemutatott algoritmusok közigazgatási szempontból releváns alkalmazásait, valamint az ezekből felépülő komplett rendszereket. Részletezni fogjuk a közbiztonsági

témájú alkalmazásokat, mint például a forgalom és a közterek megfigyelését, különböző események, anomáliák automatikus detektálását. Kitérünk ezenfelül járművek, illetve egyének azonosítására. Hasonlóan releváns téma az egyes dokumentumok automatikus felismerése, illetve feldolgozása, vagy például aláírások hitelesítése.

Szintén hangsúlyosan foglalkozunk a virtuális és kiterjesztett valóság rendszerekben való alkalmazásokkal, ezen belül is a saját kutatásunkkal, amely az adaptív tapintható kiterjesztett valóság rendszerek megvalósítását célozza meg. Foglalkozunk ezenfelül egyéb, az előbbi kategóriákba nem besorolható megoldásokkal, mint például az automatikus környezetértékelés területe.

A szerző[1]

[1] Szemenyei Márton a Budapesti Műszaki és Gazdaságtudományi Egyetem Villamosmérnöki és Informatikai Karának tanársegéde. Villamosmérnöki diplomáját a BME-n 2015-ben szerezte, kutatási területe a mesterséges intelligencia alapú számítógépes látás. Rendszeresen részt vesz az évente megrendezett RoboCup nemzetközi robotikai versenyen.

2. TÉRBELI LÁTÁS

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. Az is belátható, hogy a vetítés művelete miatt számos, számunkra fontos geometriai jellemző is torzul. Ezenfelül a térben egyenlő méretű objektumok a képen eltérő méretűek lesznek, ha a kamerától vett távolságuk más.

A vetítés során nemcsak 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 (Kató–Czúni 2011).

Látható tehát, hogy az objektumok néhány jelentős tulajdonságát lehetetlen egyetlen képből meghatározni. 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 mellet nagyjából ötszörös 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.

2.1. Kalibrációs eljárások

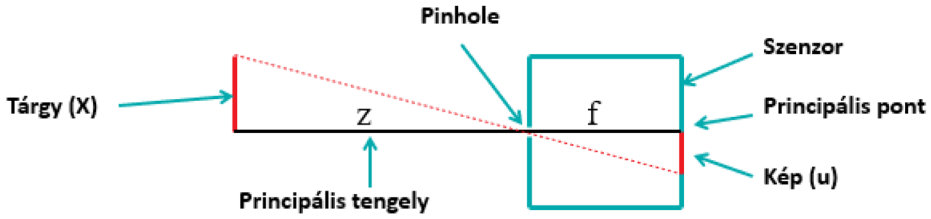
A háromdimenziós látás során tehát a feladatunk a kamera vetítése során elveszett és torzult információk rekonstrukciója. Ahhoz, hogy ezt a feladatot elvégezhessük, először meg kell érteni magát a vetítés folyamatát, vagyis fel kell állítanunk egy matematikai

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.

Jelen írás első kötetében röviden megismerkedtünk a pinhole kameramodellel (1. ábra). 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 – ahogy azt az előző kötetben is tárgyaltuk – geometriai torzítást okoz a képen. A pinhole kameramodell az alábbi egyenletek segítségével írható le (Kató–Czúni 2011):

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

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.



1. ábra: A pinhole kameramodell
Forrás: saját készítés)

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.

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \qquad \begin{pmatrix} X \\ Y \\ W \end{pmatrix} \rightarrow \begin{pmatrix} \frac{X}{W} \\ \frac{Y}{W} \end{pmatrix}, \qquad \Rightarrow \qquad \begin{pmatrix} X \\ Y \\ W \end{pmatrix} \equiv \begin{pmatrix} aX \\ aY \\ aW \end{pmatrix}$$

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 koordináta-rendszerben 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.

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

$$\begin{pmatrix} wu \\ wv \\ w \end{pmatrix} = A \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \quad ahol \quad A = \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

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 koordináta-rendszerében lettek megadva. A kamera koordináta-rendszerének 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 koordináta-rendszer, amit világ koordináta-rendszernek szokás nevezni, és a háromdimenziós pontok koordinátái általában ebben a koordináta-rendszerben vannak megadva. A világ koordináta-rendszer 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 koordináta-rendszerével megegyezzen, de bőségesen akadnak e szabály alól kivételek.

Általánosságban elmondható, hogy a világ koordináta-rendszer nem feltétlenül egyezik meg a kamera koordináta-rendszerével, így a kettő közötti transzformációt is meg kell határozni a kalibráció során. Szerencsére két koordináta-rendszer 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: extrinsic) paramétereknek nevezzük. A két pa-

ramétercsoport külön-külön meghatároz egy-egy geometriai transzformációt, kettőjük kompozíciója pedig a teljes vetítés mátrixát (*P*):

$$\vec{u} = P\vec{X} = A[R \quad t]\vec{X}$$

Ahol U a pixel, míg X a térbeli koordináták vektora. Érdemes észrevenni, hogy az előző kötetben említett geometriai torzítások hatását egyelőre nem tárgyaltuk. Ennek oka, hogy ezek a torzítások nemlineárisak, így jelentősen megnehezítik a matematikai modellezést és a becslést. A torzítások hatásával a kalibráció során külön kell foglalkoznunk.

2.1.1. Belső kalibráció

A háromdimenziós rekonstrukció elvégzéséhez első lépésként meg kell határozni a kamera vetítésének paramétereit. Ehhez bevezetésképpen először részletesen meg kellett értenünk a vetítés 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 elsősorban (de nem kizárólag) ez utóbbi belső paraméterek meghatározására szolgálnak.

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 (Szeliski 2010).

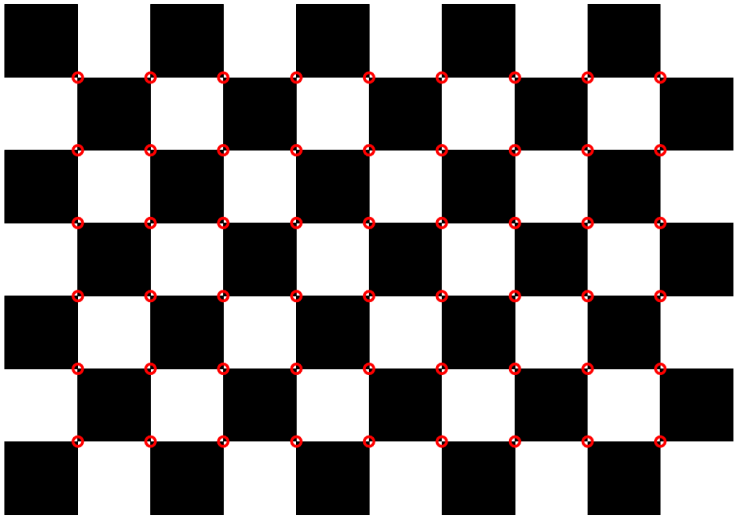
A jelen fejezetben a három- és kétdimenziós kalibrációs objektumokat használó kalibrációs módszereket fogjuk részletesebben megismerni. Mindkét esetben a kalibráció lépései megegyeznek:

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 (például négyzetek) találhatók, ahol az egyes markerek az ismétlődő formák által meghatározott sarokpontok. Kétdimenziós kalibráció esetén szinte minden esetben egy sakktáblaszerű kalibrációs objektumot (2. ábra) 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 sakktáblaszerű objektumokat, itt azonban már nehezebb a pontosságot a két objektum illesztésénél megoldani. Az ismétlődő elrendezés másik nagy előnye, hogy ha a világ koordináta-rendszert a kalibrációs objektum egy kitüntetett pontjába (például sakktábla bal felső sarka) választjuk, akkor az egyes markerek térbeli koordinátáit könnyedén előállíthatjuk bonyolult mérések nélkül.

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 végezzük, tehát minden kép esetén a kamera és a kalibrációs objektum relatív pozíciója – vagyis a külső paraméterek – más, 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 legyakrabban valamilyen sarokszerű pontok a kalibrációs objektumon, ezért célszerű valamilyen, az előző kötetben megismert sarokdetektáló módszert (például 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 (Harris–Stephens 1988).



2. ábra: Tipikus kalibrációs minta és a rajta detektált markerek
Forrás: saját készítés

Ezt követően az u=PX lineáris egyenletrendszerben már az u és az 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 *x* vektor helyén álló változó az ismeretlen, ami 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 érékét az alábbi módon:

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

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

$$\begin{matrix} up_3^T \vec{X} = p_1^T \vec{X} \\ vp_3^T \vec{X} = p_2^T \vec{X} \end{matrix} \;\rightarrow\; \begin{bmatrix} \vec{X} & \vec{0} & u\vec{X} \\ \vec{0} & \vec{X} & v\vec{X} \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = G\vec{p} = 0$$

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. (Kató–Czúni 2011)

Ezt követően a megkapott *P* vetítés mátrixot egy *A* kameramátrixra és egy *[R t]* 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. A sakktáblás kalibráció esetében azonban a *P* mátrixot nem tudjuk teljes mértékben meghatározni, így a felbontás helyett

egy másik lineáris egyenletrendszer megoldása szükséges, amely valamelyest több számítást igényel.

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:

$$E_G = \sum \|u_i - f(X_i, P)\|^2$$

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, amelyekről a 3. fejezetben tárgyalunk majd bővebben (Goodfellow–Bengio–Courville 2016).

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.

2.1.2. Kamerarendszerek kalibrációja

Amint azt a fejezet bevezetőjében megállapítottuk, a háromdimenziós rekonstrukcióhoz több kamerára, vagy legalábbis több különböző pozícióból elkészített felvételre van szükség. Ezenfelül 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.

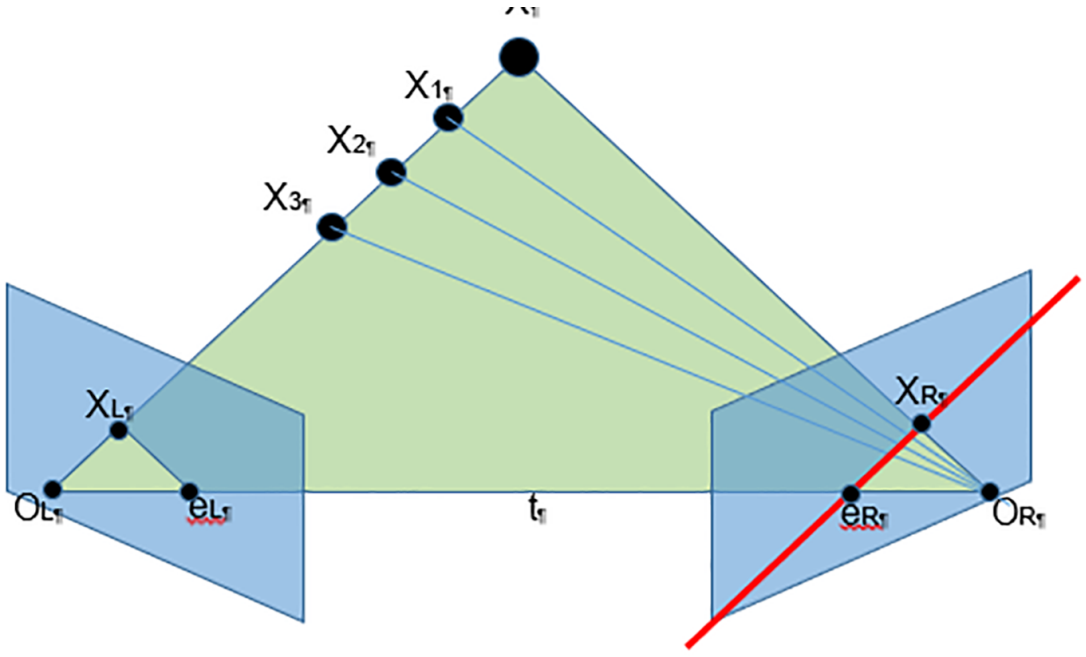
A lehetséges módszerek feltárásához azonban érdemes felismerni, hogy a feladat természete és következésképp a nehézsége nagymértékben függ az elrendezéstől. A korábbi fejezetekben is megkülönböztettük azt a két esetet, amikor két kamera készít külön pozícióból két felvételt, és amikor egy kamera mozdul 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 használatakor 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 látszódjék. Emlékezzünk vissza, hogy az összes kalibrációs objektumot használó eljárás esetén nemcsak a kamerák belső paraméterei, hanem a külső paraméterek (vagyis a kalibrációs objektum és a kamera közti transzformáció) is előálltak minden kalibrációs képhez (Kató–Czúni 2011).

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.

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.



3. ábra: A sztereó elrendezés
Forrás: saját készítés

Ehhez azonban először meg kell vizsgálnunk magát a geometriai elrendezést. Az egyszerűség kedvéért szorítkozzunk az úgynevezett sztereó elrendezéshez (3. ábra), 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 kamera-középpont 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 (Kató–Czúni 2011).

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:

$$x_L^T(t \times R^T x_R) = x_L^T([T_x]R^T)x_R = x_L^T E x_R = 0$$

Ahol ($T_{x)}$ 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 koordináta-rendszerébe kellett forgatni, ugyanis minden vektort ugyanabban a koordináta-rendszerben 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 koordináta-rendszerében értelmezett vektorok. A kettő között az áttérést a kameramátrix adja meg:

$$x_L = A_L^{-1} u_L$$

Ezt behelyettesítve:

$$u_L^T A_L^{-T}([T_x]R^T)A_R^{-1}u_R = u_L^T F u_R = 0$$

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.

A következő fontos lépés az F meghatározása mérésekből. Ennek alapvető módszere, hogy mindkét képen természetes markereket keresünk, majd ezeket valamilyen jellemző leíró (például 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 – hét, illetve nyolc pontpár segítségével számolják ki F értékét (Szeliski 2010).

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:

$$Af = \begin{bmatrix} u_{L1}u_{R1} & u_{L1}v_{R1} & u_{L1} & v_{L1}u_{R1} & v_{L1}v_{R1} & v_{L1} & u_{R1} & v_{R1} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_{LN}u_{RN} & u_{LN}v_{RN} & u_{LN} & v_{\ln}u_{RN} & v_{LN}v_{RN} & v_{LN} & u_{RN} & v_{RN} & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} = 0$$

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 (Szeliski 2010).

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.

2.2. Rekonstrukciós eljárások

Az előző fejezetekben megismerkedtünk a képalkotás geometriájának alapvető összefüggéseivel, valamint a kalibrációs eljárások legfontosabb részleteivel. Mindezek a háromdimenziós rekonstrukció alapvető fontosságú részletei, ezenfelül bármilyen rekonstrukciós rendszer felépítésének első, megalapozó lépései. A jelenlegi alfejezet témája azonban már a konkrét rekonstrukció kivitelezése és az ehhez szükséges számítógépes látás algoritmusok ismertetéséről szól.

A háromdimenziós rekonstrukció elvégzését alapvetően négy lépésre oszthatjuk, amelyek közül kettőt már a korábbi fejezetekben tárgyaltunk. Az első ismertetett 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.

2.2.1. Megfeleltetések keresése

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 (4. ábra) 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.

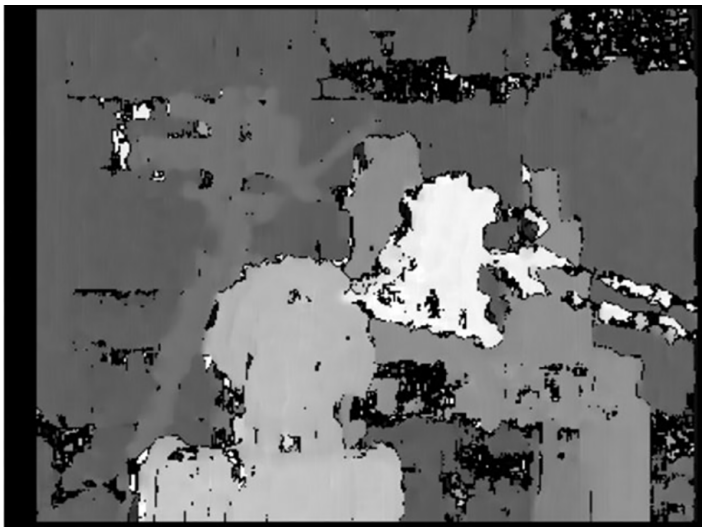
Amennyiben a kép minden pixeléhez meghatározzuk a diszparitás értéket, akkor egy szürkeárnyalatos képet kapunk, amelyet diszparitás képnek hívunk. Ez a kép könnyen értelmezhető az emberi szem számára is, hiszen a diszparitás értéke alapvetően a kamerától vett távolságtól függ. Feltehetően életében egyszer 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 közötti különbség. Innen könnyű belátni, hogy minél nagyobb a diszparitás érték, annál közelebb van az adott pixelhez tartozó tárgy a kamerához (Szeliski 2010).

A diszparitás meghatározására számos módszer létezik, amelyeket 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.

4. ábra: A tsukuba sztereó képpár

Forrás: tsukuba sztereó adatbázis http://www.cvlab.cs.tsukuba.ac.jp/dataset/tsukubastereo.php)

A sűrű diszparitás módszerek egyik legfontosabb családja a block mathing (BM) algoritmusra épülő eljárások (Chan–Panchanathan 1993). A block matching a számítógépes látás egyik alapvető algoritmusa, amelynek számos alkalmazása létezik, amelyekben gyakran valamelyest eltérő néven említik. Az algoritmus lényege, hogy a vizsgált pixel egy környezetét véve végighalad 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.



5. ábra: A block matching algoritmus eredménye
Forrás: saját készítés

Érdemes megjegyezni, hogy a block matching algoritmus az egyes pixelek diszparitását külön-külön, 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 (5. ábra) meglehetősen nagy zajjal terhelt.

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 (Hirschmüller 2008). 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 a 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.

6. ábra: A BP algoritmus eredménye
Forrás: saját készítés

A diszparitás számolásának másik kiváló módszere a belief propagation (BP) módszer (Sun–Zheng–Shum 2003). Ennek az eljárásnak a lényege, hogy az egyes diszparitások értékét valószínűségi változóként értelmezi. Minden pixel ismeri a saját diszparitásának eloszlásfüggvényét, amelynek a maximum helye az adott pixel „hite”. Ezt követően minden pixel üzenetet küld az összes szomszédjának: ezek az üzenetek voltaképpen a szomszédok diszparitására vonatkozó valószínűségi eloszlások. Minden pixel úgy gondolja, hogy a szomszédjainak hozzá hasonló diszparitása van, így ez az eloszlás célszerűen egy normális eloszlás, ahol a várható érték az adott pixel hite.

Miután minden pixel kapott egy üzenetet az összes szomszédjától, a korábbi és a kapott eloszlások felhasználásával minden pixel egy új eloszlást konstruál, az új hit pedig ennek a maximum helye lesz. Ezt a folyamatot addig ismételjük, amíg a diszparitás kép egy konszenzusállapothoz nem konvergál. Diszparitásképek esetére a konvergencia bizonyítottan garantált. Az algoritmus működése során a kezdeti eloszlást könnyedén generálhatjuk például a block matching módszer eredménye alapján: Ez a metódus minden diszparitáshoz előállít egy hibaértéket, 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 diszparitásugrásokat jelölik. A belief propagation algoritmus lényegesen jobb minőségű képet (6. ábra) szolgáltat az SGBM-módszerhez képest, azonban mindezt jelentős számításigény árán. A legtöbb rendszerben a BP-eljárást érdemes inkább grafikus célhardver segítségével gyorsítani.

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 változatait az első kötetben részletesen ismertettem. A diszparitásképzéshez hasonlóan az optikai áramlásnak is léteznek sűrű, illetve ritka változatai, az utóbbi 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 az első kötetben ismertetett lokális képjellemző leíró módszereket párosításra használni, habár ezeket a ritka eljárásokat gyakrabban szokták a kalibrációhoz szükséges pontpárok keresésére használni.

2.2.2. Sztereó rekonstrukció

A háromdimenziós rekonstrukció utolsó lépése a reprojekció, amelynek során a kép pontjait visszavetítjük a háromdimenziós térbe. 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 koordináta-rendszerbe. Ebben a közös viszonyítási rendszerben az ugyanazon térbeli ponthoz tartozó vetítési sugarak az eredeti térbeli pont pozíciójában fogják egymást metszeni.

Az imént ismertetett módszer a háromszögelés (7. ábra) 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 négy-

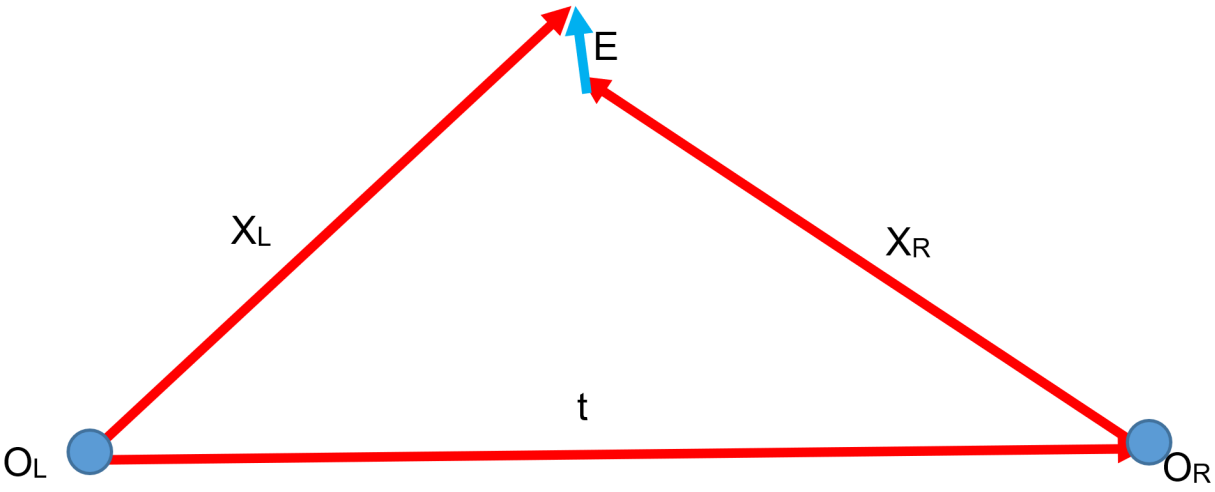
zetek módszerének segítségével meghatározni. Kezdetnek felírjuk a vetítés egyenletét, majd abból kifejezzünk a képpont *u koordinátáját:* (Szeliski, 2010)

$$\begin{pmatrix} w_1u_1 \\ w_1v_1 \\ w_1 \end{pmatrix} = \begin{bmatrix} p_{11}^T \\ p_{12}^T \\ p_{13}^T \end{bmatrix} \vec{X} \rightarrow u_1 = \frac{w_1u_1}{w_1} = \frac{p_{11}^T\vec{X}}{p_{13}^T\vec{X}}$$

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:

$$\begin{matrix} u_1p_{13}^T\vec{X} = p_{11}^T\vec{X} \\ v_1p_{13}^T\vec{X} = p_{12}^T\vec{X} \end{matrix} \rightarrow \begin{bmatrix} p_{11}^T\,\vec{0} & -u_1p_{13}^T \\ \vec{0}\,p_{12}^T & -v_1p_{13}^T \end{bmatrix} \vec{X} = 0$$

Itt egy csupa ismert értékből álló mátrixot szoroz balról egy csupa ismeretlen vektor, 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 legkiseb 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.



7. ábra: A háromszögelés elve

Forrás: saját készítés

Számos térbeli számítógépeslátás-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, amelynek segítségével az eredeti térbeli pozíciók a

$$X = Q \begin{bmatrix} u \\ v \\ d \end{bmatrix}$$

formában adódnak, ahol u és v az X pont egyik képen megtalált pozíciója, d pedig a diszparitás értéke.

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.

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 (Szeliski 2010).

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.

2.2.3. Többkamerás rekonstrukció

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. Ezenfelül 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 (Szeliski 2010).

Valamivel érdekesebb az egy mozgó kamerát alkalmazó rekonstrukció esete. Ugyan itt valójában nehezen beszélhetünk többkamerás rekonstrukcióról, azonban mozgókamerás 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 (Wu 2013) (az angol Structure from Motion – mozgásalapú struktúra) és a SLAM (Taketomi–Uchiyama–Ikeda 2017) (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.

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. Jellemződetektálás
2. Az előző kép jellemzőivel párosítás
3. Kamerapóz becslése
4. 3D pontok koordinátáinak becslése
5. Csoportigazítás (Bundle Adjustment)

Külön kiemelendő a csoportigazítás művelete (Wu–Agarwal–Curless–Seitz 2011), 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 összeségéből áll össze. 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, amelyek közül az első, hogy a relatív elmozdulást nemcsak az előző, hanem a 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, amelynek során érzékeljük, ha a kamera visszaért egy korábban meglátogatott pozíció közelébe, és ebben az esetben a korábbi pozícióban készített képkockához képest becslünk pozíciót. Általánosságban is lehetséges az a megoldás, hogy az előző képkockák mellett a már elkészült térképrészlethez képest is megkíséreljük a lokalizációt.

2.3. Pontfelhők feldolgozása

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 módszereiről 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.

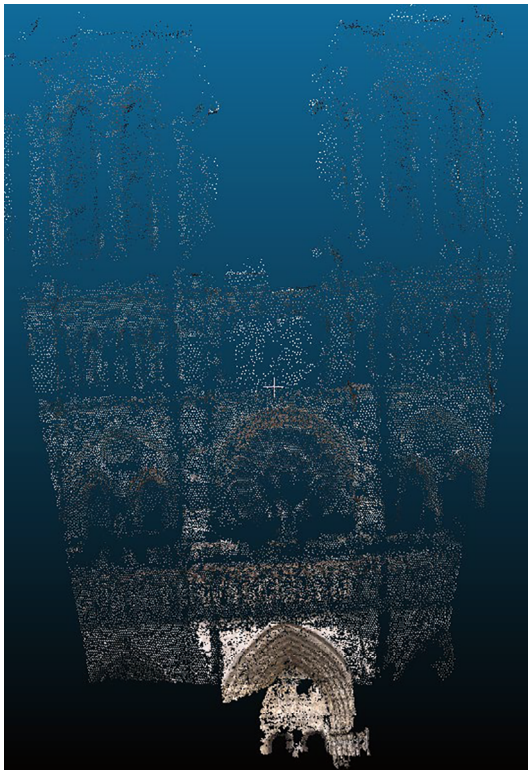
Mielőtt azonban belekezdnénk az egyes módszerek ismertetésébe, egy alapvető kérdést szükséges még tisztáznunk. Az ugyanis nem nyilvánvaló, hogy a térbeli struktúrát milyen formában reprezentáljuk. A hagyományos számítógépes látás egy kétdimenziós képet egy kétdimenziós rácson tárol, ahol a rács minden 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 általgos

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. Biztatom 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 struktúrák alapvető tárolóstruktúrája a pontfelhő (8. ábra), 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 legyakoribb 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.



8. ábra: Példa pontfelhőre: A Notre Dame háromdimenziós rekonstrukciója turistafotók segítségével
Forrás: saját készítés

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 az háromszögháló (mesh) használata, ahol a komplex felületeket háromszög alakú építőelemekkel közelítenek. Ez utóbbi módszer elsősorban a számítógépes grafikában elterjedt.

2.3.1. Szűrések

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 voxelszűrő. A szűrő működése analóg a képeknél megismert átlagoló szűrőhöz: a dobozszűrő egy háromdimenziós ablakkal végigpásztázza a pontfelhőt, és minden lépésben a doboz területébe eső pontokat az átlagukkal helyettesíti. A dobozszűrő 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 dobozszűrő segítségével azonban ezt kordában tudjuk tartani.

Léteznek azonban olyan hibás pontok a pontfelhőkben, amikkel szemben a dobozszűrő 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 outliernek 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.

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 (Bentley 1975). A kd-fa a pontfelhő pontjait egy gráfban helyezi el, ahol egy gyökércsomópont 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.

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.

2.3.2. 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 kismonográfia első kötetében megismerkedtünk számos, képeken használható szegmentálási módszerrel, többek között a régióalapú módszerekkel, mint a régiónövesztés és -szeletelés módszer, a klaszterezésalapú eljárásokkal, mint a k-means és a MoG, valamint a gráfvágáson 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 pixelértékek hasonlósága volt, pontfelhők esetében gyakrabban használnak a térbeli összetartozáson alapuló mércéket.

Létezik azonban egy algoritmus, amelyet ritkán használnak képekre, pontfelhők szegmentálása esetében azonban az egyik legnépszerűbb eljárásnak számít. Ez az algoritmus a RANSAC (Fischler–Bolles 1981) (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 nagyszámú véletlen jelöltet állít elő. Ezt követően megvizsgálja, hogy az egyes jelöltekre a teljes ponthalmazból hány pont illeszkedik rá. Az egy adott jelöltre illeszkedő pontokat inliernek hívjuk. A 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 (Schnabel–Wahl–Klein 2007).

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. Általánosságban is elmondható, hogy a mintavételezés és kiértékelés módszereit érdemes az adott feladatra szabni.

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 az 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 valós idejű algoritmus.

2.3.3. 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 jellemzőit, 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 robusztusok a különböző transzformációkra.

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 (Rusu–Blodow–Beetz 2009) (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.

Globális, egy egész objektumot leíró jellemzőre rendkívül jó példa a GASD (Lima & Teichrieb, 2016) (Globally Aligned Spatial Distribution – globálisan illesztett térbeli eloszlás) módszere. Ennek a módszernek a lényege, 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 MxMxM 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.

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 több, egymással részleges átfedésben lévő pontfelhőrészlet 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.

3. INTELLIGENS LÁTÓRENDSZEREK

Az előző fejezet során részletesen bemutattuk a számítógépes látás egyik komplex alkalmazási területét, a háromdimenziós látást. A jelenlegi fejezet témája egy szintén jelentős terület, amely a számítógépes látás és a mesterséges intelligencia tudományágának ötvözéseként állt elő. Az intelligens látórendszerek egy olyan újszerű és rohamosan fejlődő kutatási területté váltak az utóbbi évtizedben, amely semmilyen, a számítógépes látáshoz kapcsolódó diszkusszió során nem hagyható figyelmen kívül. Kijelenthető, hogy az utóbbi évtizedben a számítógépes látás jelentős kutatási eredményeinek és áttörést jelentő ipari alkalmazásainak túlnyomó többsége erről a területről érkezett.

A jelenlegi fejezetben első lépésként áttekintjük a gépi tanulás alapjait, illetve alapvető nehézségeit és problémáit. Ezt követően bemutatjuk a gépi tanulás két fontos fajtáját: a felügyelt, illetve a nem felügyelt tanulás alapjait, illetve releváns algoritmusait. Természetesen a gépi tanulás számos algoritmusa közül a számítógépes látás szempontjából releváns részeket mutatjuk majd be. A tárgyalás során a legnagyobb hangsúly az úgynevezett mély tanulás (deep learning) (Goodfellow–Bengio–Courville 2016) megoldásaira helyeződik, tekintve, hogy ez az utóbbi évek egyik legfontosabb technológiája.

3.1. Gépi tanulás alapjai

A számítógépes látás területének alapvető célja magas szintű információk kinyerése a képekből. Ennek legegyszerűbb formája a már korábban ismertetett osztályozás, 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 többfajta 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 maszkképet 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 objektumszegmentálás feladatáig.

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

Ezen feladatok során azonban számos nehézségbe ütközünk, amelyek közül az egyik legfontosabb az előző kötetben már említett szemantikus gát, vagyis a látszólag áthidalhatatlan különbség a képek digitális reprezentációja és az emberi értelmezés között. Ez a gát teszi kvázi lehetetlenné a bonyolultabb számítógépeslátás-problémák egyszerű algoritmussá történő megfogalmazását. Ezenfelül jelentős problémát jelentenek még az egyes osztályokon belüli variációk és a különböző képi transzformációk, amelyek a pixelértékeket jelentősen megváltoztatják, a kép szemantikus jelentését viszont alig vagy egyáltalán nem befolyásolják. Ilyen transzformációk a megvilágítás változásai, egyes objektumok deformációi vagy részleges takarása.

Éppen a fenti problémák miatt adja magát a felvetés, hogy az emberi intelligencia képességeit próbáljuk meg valamilyen módon a számítógépes látás módszereibe beleültetni, ezáltal 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 feketedoboz-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.

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:

$$y = f(\mathrm{x}; \vartheta)$$

Minden tanító algoritmushoz tartozik egy költségfüggvény (gyakran nevezik még hiba- vagy veszteségfüggvénynek), amely a tanuló algoritmus kimenetéhez hozzárendel egy hibaértéket, amelynek segítségével az algoritmus teljesítményét tudjuk értékelni. Ezenfelül 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, amelyek 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 (James–Witten–Hastie–Tibshirani 2009).

A gépi tanulás módszereit számos fontos szempont szerint lehetséges csoportosítani, amelyek 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.

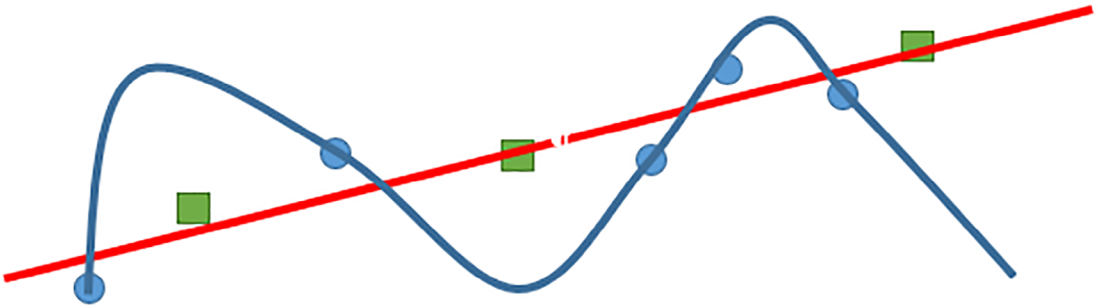
Létezik azonban felügyelet nélküli tanulás, amikor a tanító adathalmaz csak bemeneti értékekből áll, az elvárt kimenetet egyáltalán nem ismerjük. Ilyen esetekben azt várjuk el a tanuló algoritmustól, hogy képes legyen valamilyen belső struktúrát találni az adathalmazban, és ezáltal azt kompakt módon leírni. 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 meghoznia, 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 (például sakk, go, számítógépes játékok), valamint a különböző járműirányítási feladatok (James–Witten–Hastie–Tibshirani 2009).

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, a teljesítmény növelhető. Egy modell komplexitását számos módon lehet növelni: a bemeneti változók és a paraméterek számának növelése két nyilvánvaló módszer. Ezenfelül 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, amelyeket a tanítás során nem látott, akkor hasonlóan nagy 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) (Goodfellow–Bengio–Courville 2016).



9. ábra: Az overfitting jelensége szemléletesen. A bonyolultabb, kék görbe jobban illeszkedig a kék színnel jelölt tanító adatokra, az új mintákra adott válasza azonban teljesen hibás, míg egy egyszerűbb modell mindkét esetben jól teljesít

Forrás: saját készítés

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 (9. ábra) 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.

A túlillesztés jelenségét csupán a tanítási hibából lehetetlen diagnosztizálni, ezért minden tanítás esetén alapvető, hogy a tanító adatbázis egy részét (10-20 százalékát) félretesszük, és nem használjuk tanításra, hanem időnként validáljuk az algoritmust ezekkel az adatokkal. Ha a validációs adatbázison elért pontosság lényegesen rosszabb, mint a tanító adatbázison, akkor túlillesztés történt. Ily módon a validáció segítségével

képesek vagyunk mérni a túlillesztés mértékét. Maga a jelenség elkerülhető vagy mérsékelhető az által, hogy az optimalizáláshoz használt hibafüggvényt kiegészítjük egy, a komplexitást büntető taggal. Ez a regularizációnak nevezett eljárás biztosítja, hogy kompromisszumot találjunk a tanítási hiba minimalizálása és a komplexitás kordában tartása között (Goodfellow–Bengio–Courville 2016).

Egy utolsó megfontolandó dolog a tanuló osztályozó algoritmusok teljesítményének a mérése. A legtöbb esetben egyszerűen a helyes osztályozások arányát szoktuk használni, ez azonban félrevezető lehet. Előfordulhat ugyanis, hogy az egyik osztályból lényegesen több van, mint a másikból. Ebben az esetben az algoritmus nagy pontosságot tud elérni csupán azáltal, hogy az összes bemenetre a gyakrabban előforduló osztályt választja. Éppen ezért érdemes általában két mérőszámot használni, melyek közül a leginkább elterjedt a pontosság és az emlékezés (precision és recall) mérőszámok.

A pontosság azt fejezi ki, hogy a tanuló algoritmus által pozitívnak kiválasztott elemek (vagyis az éppen vizsgált osztályba tartozó) hány százaléka volt valóban pozitív. Az emlékezés pedig azt fejezi ki, hogy a ténylegesen pozitív elemek hány százalékát választotta az algoritmus pozitívnak. A jó osztályozáshoz ezeknek a mérőszámoknak együtt kell kielégítőnek lenniük. Érdemes még megjegyezni, hogy többértékű osztályozás esetében az úgynevezett keverési mátrixot szokás vizsgálni, amely voltaképpen ezen mérőszámok kiterjesztése.

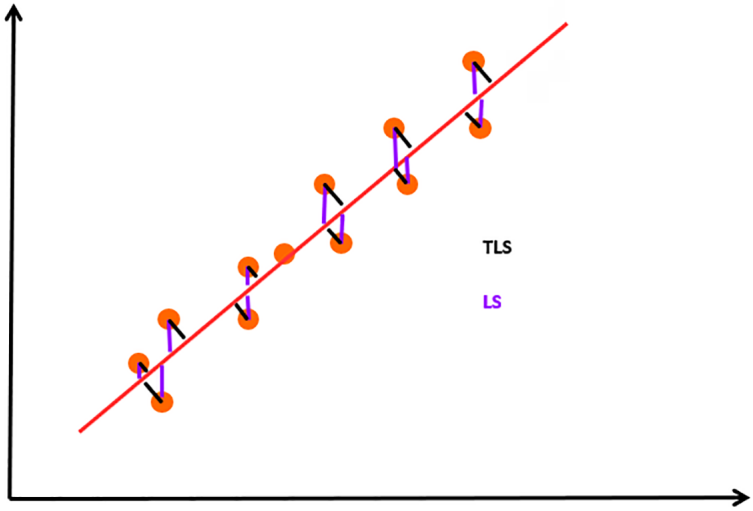
3.2. Felügyelt tanulás

Miután a gépi tanulás alapjaival megismerkedtünk, elkezdhetjük a számítógépes látás szempontjából releváns módszerek tárgyalását. Első körben a felügyelt tanulás néhány releváns algoritmusával ismerkedünk meg röviden. A jelenlegi alfejezet hátralévő részében pedig a mély tanulás területét tárgyaljuk részletesen. A felügyelt tanulás egyik legegyszerűbb algoritmusa a korábbi kötetben röviden már tárgyalt legközelebbi szomszéd (kNN) eljárás. Ezt a módszert azonban a számítógépes látásban szinte soha nem használjuk, mivel a képek közötti pixelenkénti eltérés és a képek szemantikus hasonlósága között nincs összefüggés.

Egy másik alapvető tanuló algoritmus a lineáris regresszió, vagy lineáris legkisebb négyzetes becslés (LS) módszere. Ezt a módszert gyakran használják tanuló algoritmusként, valamint paraméterbecslő algoritmusként is. 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 (10. ábra). A lineáris regresszió modellje az alábbi:

$$X\vartheta = Y$$

Ahol az 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 nemcsak különböző változók, hanem akár egyetlen változó különböző hatványai is szerepelhetnek. Ekkor nem egyenest, hanem valamilyen polinomot illesztünk az adathalmazra, mely esetben polinomiális regresszióról beszélhetünk.



10. ábra: Az LS-módszer grafikus szemléltetése. A cél a fekete szakaszokkal jelölt hibák minimalizálása. A TLS-módszer esetén a hibatagok nem a függőleges irányába, hanem az illesztett egyenesre merőleges irányban értelmezhetők.
Forrás: saját készítés

A lineáris regressziónak létezik egy valamelyest különböző formája is, amikor a fenti egyenletben szereplő 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 (James–Witten–Hastie–Tibshirani 2009).

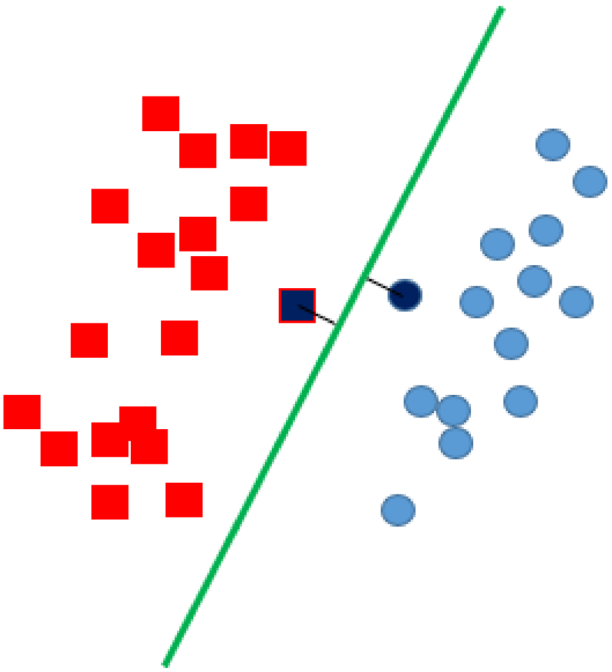
A lineáris megközelítést az osztályozás módszereire is ki lehet terjeszteni, ezt azonban úgy lehet elképzelni, hogy az egyes osztályokat egy egyenessel (hipersíkkal) próbáljuk meg egymástól elválasztani. Könnyen belátható azonban, hogy általában végtelen sok ilyen sík létezik, így valamilyen módon jó 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 (11. ábra). A síkhoz legközelebbi tanító adatot szupport vektornak, míg annak a síktól való távolságát résnek (margin) nevezzük.

A számítógépes látásban népszerű algoritmus az úgynevezett SVM (Cortes–Vapnik 1995) (Support Vector Machine) algoritmus, amely az osztályokat maximális réssel elválasztó hipersíkot keresi meg. Az SVM döntésfüggvénye az alábbi módon írható fel:

$$y_{pred}(x) = \sum_{i}^{N} \alpha_i \, y_i K\left(x^{(i)}, x\right)$$

Ahol *N a tanító adatok száma,* y_i és $x^{(i)}$ az i-edik tanító adat be- és kimenete, 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.



11. ábra: Bináris osztályozás. Sötétkék színnel jelöltek a szupport vektorok
Forrás: saját készítés

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.

$$K(x_1,x_2) = x_1^T x_2$$

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).

$$K(x_1,x_2) = (x_1^T x_2 + c)^k$$

$$K(x_1,x_2) = e^{-\gamma\|x_1 - x_2\|^2}$$

Ahol c, k és y a módszer komplexitását kontrolláló hiperparaméterek.

3.2.1. Képosztályozás

A felügyelt tanulás alapvető módszereinek rövid bemutatása után a jelenlegi fejezettel kezdődően a mély tanulás alapú látórendszerek területének módszereit fogom részletesen bemutatni. A mély tanulórendszerek 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:

$$f(x, W) = Wx$$

Ahol *x* a bemenet, *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). A módszer alapvető kérdése azonban, hogy hogyan lehet a súlyok értékét úgy meghatározni, hogy az osztályozás minél pontosabb legyen, valamint hogy milyen költségfüggvény segítségével mérhető jól az algoritmus teljesítménye (Goodfellow–Bengio–Courville 2016).

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 osztályozást végző modell között. Éppen ezért a modell kimenete és az adott tanító adathoz 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 (Goodfellow–Bengio–Courville 2016). Ennek a hibafüggvénynek az alapelve, hogy definiálunk egy Δ 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 (12. ábra) előíró kritériumként. Az SVM-hiba formálisan a következő:

$$L_i = \sum_{j \neq corr} max\left(0, s_j - s_{corr} + \Delta\right), ahol\ s = f(x_i, W)$$

Ahol S_j a j-edik, S_{corr} pedig a helyes osztály jóságértéke. Létezik egy másik gyakorlatban elterjedt hibafüggvény, amelyik a geometriai szemlélet helyett inkább a valószínűség-számítás 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:

$$H(p) = \sum_i p_i \frac{1}{\log p_i} = -\sum_i p_i \log p_i$$

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.

$$H(p, q) = \sum_i p_i \frac{1}{\log q_i} = -\sum_i p_i \log q_i$$

$$KL(p \parallel q) = H(p, q) - H(q)$$

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 pontosan egy legyen. A SoftMax függvény az alábbi módon írható fel:

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:

$$L_i = H(p_i, q_i) = -\sum_k p_{k,i} \log q_{k,i} = -log q_{corr,i}$$

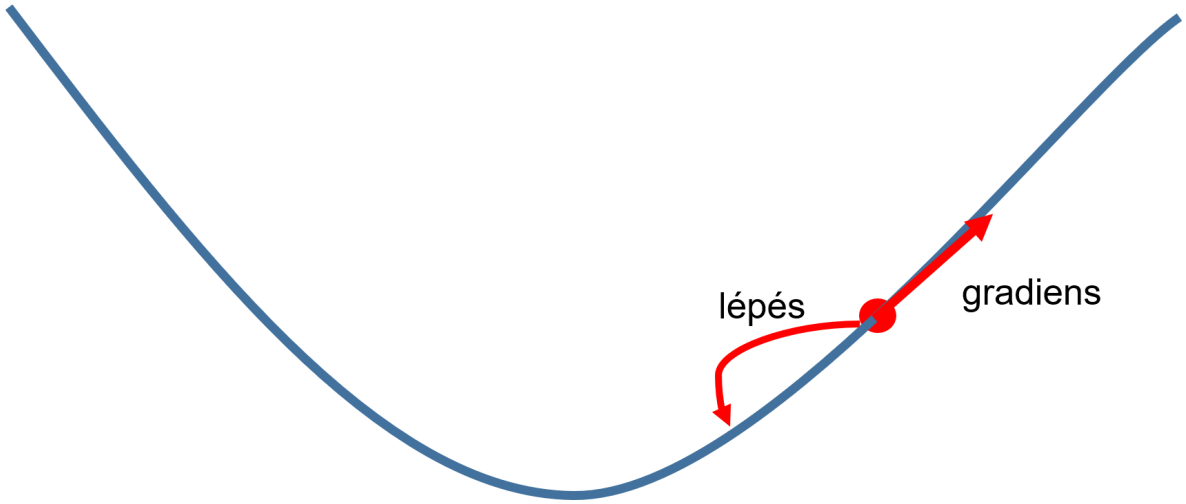
Ahol $p_{k,i}$ és $q_{k,i}$ az i-edik tanító adat k-adik osztályhoz tartozó előírt és becsült valószínűségei, $q_{corr,i}$ pedig a helyes osztály becsült valószínűsége. A keresztentrópia költségfüggvény egyik előnye, hogy nehezen értelmezhető „jóság” értékek helyett valószínűség jellegű értékekkel dolgozik, 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 többi tanító adat helyes osztályozására fordítsa. A gyakorlatban a két költségfüggvény közti különbség azonban alig kimutatható.

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 ezenfelül még az úgynevezett elasztikus regularizáció, amikor a kétfajta norma súlyozott átlagát használják. A végső hibafüggvény a következőképp adódik:

$$L = \frac{1}{N} \sum_i L_i + \lambda R(W)$$

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 (GOODFELLOW–BENGIO–COURVILLE 2016).



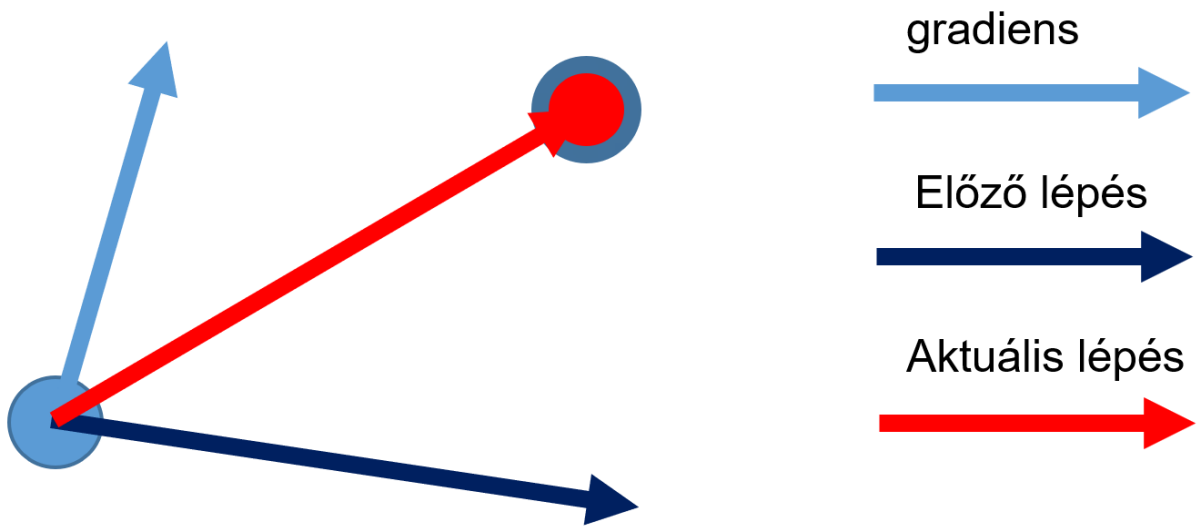
13. ábra: A gradiens módszer szemléltetése egy dimenzióban
Forrás: saját készítés

A jelenlegi alfejezet utolsó kérdése az imént bemutatott hibafüggvények minimalizálására szolgáló optimalizálási módszerek tárgyalása. A perceptron súlyainak optimális értékére nem létezik zárt alakú megoldás, így iteratív optimalizálásra lesz szükség. Szerencsére a modell és a költségfüggvény is deriválható, így alkalmazhatunk gradiensalapú módszereket. Ha kiszámoljuk a hibafüggvény súlyok szerinti deriváltját (más szóval a súlyok gradiensét), akkor megkapjuk azt, hogy hogyan kellene a súlyoknak megváltoznia ahhoz, hogy a hibafüggvény a lehető leggyorsabban növekedjen. Ha azonban a súlyokat a gradienssel ellenkező irányba változtatjuk, az a leggyorsabb csökkenés iránya lesz (13. ábra). Ezt a lépést egyfajta „fordított hegymászóként” ismételve egy idő után lokális minimumba jutunk.

Vegyünk azonban észre, hogy mivel a teljes hibát szeretnénk minimalizálni, ezért minden egyes lépéskor ki kell az értékelni az egész tanító adatbázis hibáját. Mivel a gradiens módszer aránylag sok lépés után konvergál csak, azért ez nem praktikus. Éppen ezért a tanító adatbázist egyenlő méretű, véletlenszerűen kiválasztott részhalmazokra (minibatchekre) osztjuk, és minden egyes minibatch után végzünk egy lépést. Ezt a módszert sztochasztikus gradiens módszernek (SGD – Stochastic Gradient Descent) nevezzük (Ruder 2016). A minibatchek mérete általában a kettő hatványa szokott lenni, implementációs okokból. Érezhető persze, hogy az egyetlen minibatchre 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:

$$W_{k+1} = W_k - \frac{\alpha}{N_{MB}} \sum_{i}^{N_{MB}} \nabla L_i$$

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

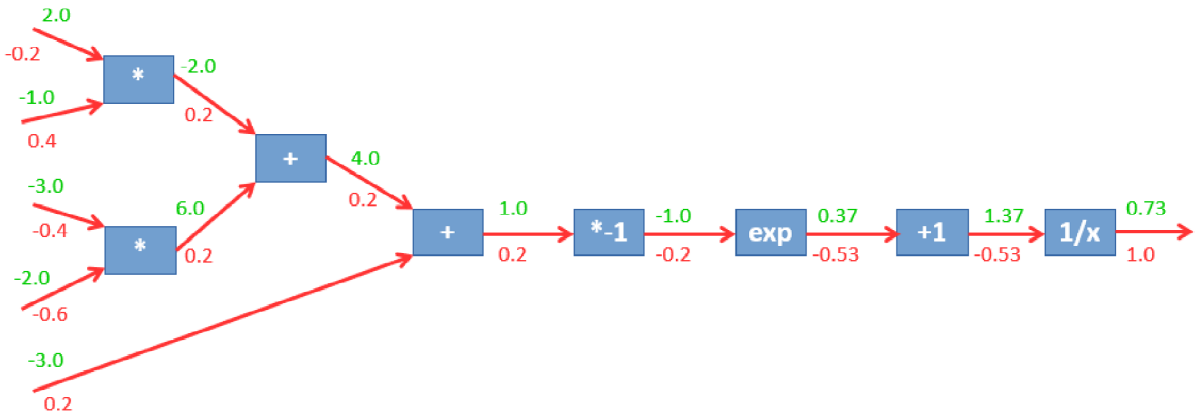


14. ábra: Az egyszerű momentum módszer szemléltetése
Forrás: saját készítés

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 (14. ábra) 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 (RUDER 2016). Ez a súly alkalmazástól függően általában a [0,1-0,9] tartományban mozog.

3.2.2. Mély neurális hálók

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 neuronmodelleket 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.



15. ábra: Példa számítási gráfra. Zöld színnel az aktivációk, pirossal a kimenetek adott érték szerinti deriváltjai látható
Forrás: saját készítés

Az egyetlen kérdéses lépés a hibafüggvény súlyok szerinti deriváltjának számítása. Egy neurális háló elképzelhető mint egy számítási gráf, ahol a gráf egyes csomópontjai egyszerű, analitikusan deriválható függvényeket implementálnak (15. ábra). 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 (GOODFELLOW–BENGIO–COURVILLE 2016). 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.

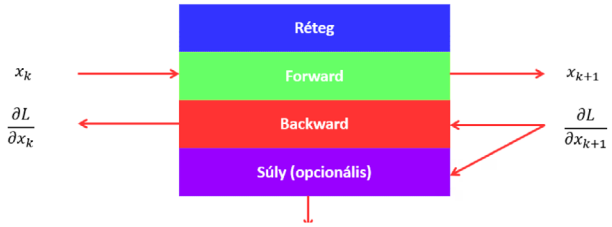
$$\frac{\partial L}{\partial x} = \frac{\partial f}{\partial x}\frac{\partial L}{\partial f}, \text{és } \frac{\partial L}{\partial W} = \frac{\partial f}{\partial W}\frac{\partial L}{\partial f}$$

Ahol *L* a hibafüggvény, *f* és *x* az adott réteg ki- és bemenete, *W* pedig a réteg bemenete és 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.

Érdemes megjegyezni, hogy a hátraterjesztés művelete (Le Cun, 1988) 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. A felhasználásoknak itt még nincs vége: a backpropagation felhasználható arra, hogy megkapjuk azt a képet, ami a háló egy kiszemelt neuronját a lehető legerősebben aktiválja. Ezzel a vizualizációs módszerrel választ kaphatunk arra a kérdésre, hogy mit csinál egy adott neuron. Ilyen jellegű kérdések megválaszolása a gépi tanuló algoritmusok feketedoboz-jellege miatt általában rendkívül nehéz.

A backpropagation algoritmus alkalmazásai során fontos felfedezés volt, hogy egy tipikus számítógépes látásban használt neurális háló esetén lehetséges helyesen osztályozott képeken olyan minimális, emberi szem által észrevehetetlen változásokat generálni, amelyek 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.

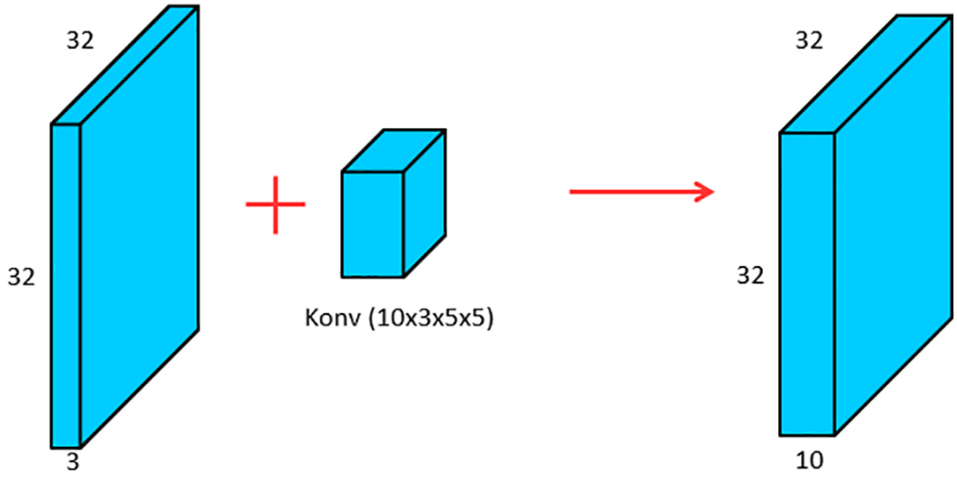
Érdemes azt is észrevenni, hogy a neurális háló elemeinek nem feltétlenül kell a korábbi fejezetben megismert perceptron függvényeknek lenniük. A valóságban a neurális hálózatok számos különböző fajta rétegből állnak, amelyeknek közös tulajdonságuk, hogy deriválható függvényeket valósítanak meg. Saját magunk is könnyedén készíthetünk új típusú rétegeket egészen addig, amíg az előre- és hátraterjesztés részfeladatait elvégző függvényeket megvalósítjuk (16. ábra).



16. ábra: Általános rétegmodell egy mély neurális hálóban
Forrás: saját készítés

A korábban ismertetett lineáris neuronmodell 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.

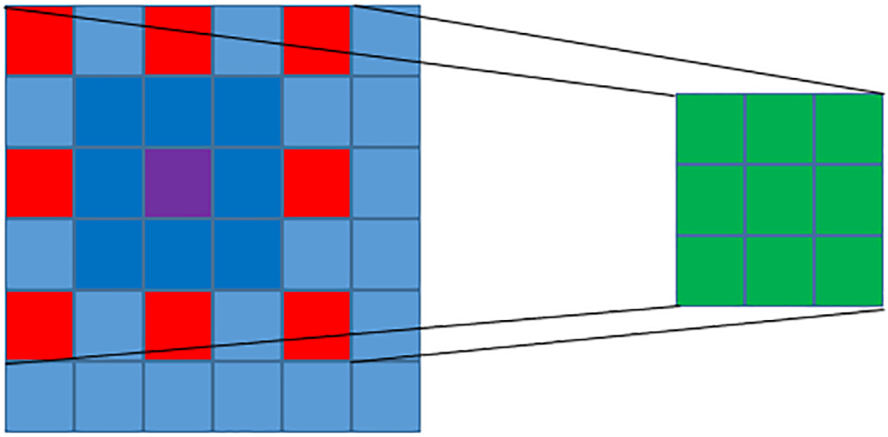
Ezekre a problémákra ad megoldást a konvolúciós réteg (17. ábra), amely nevéből adódóan a korábbi kötetben ismertetett konvolúciós szűrőkhöz hasonlóan működik. Egy konvolúciós réteg a bemeneti (általában 1-3 csatornás) képen N darab különböző konvolúciós szűrőt futtat végig, amelynek eredménye egy N-csatornás 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.



17. ábra: Egy konvolúciós réteg struktúrája

Forrás: saját készítés

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 (18. ábra) 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 (Goodfellow–Bengio–Courville 2016).



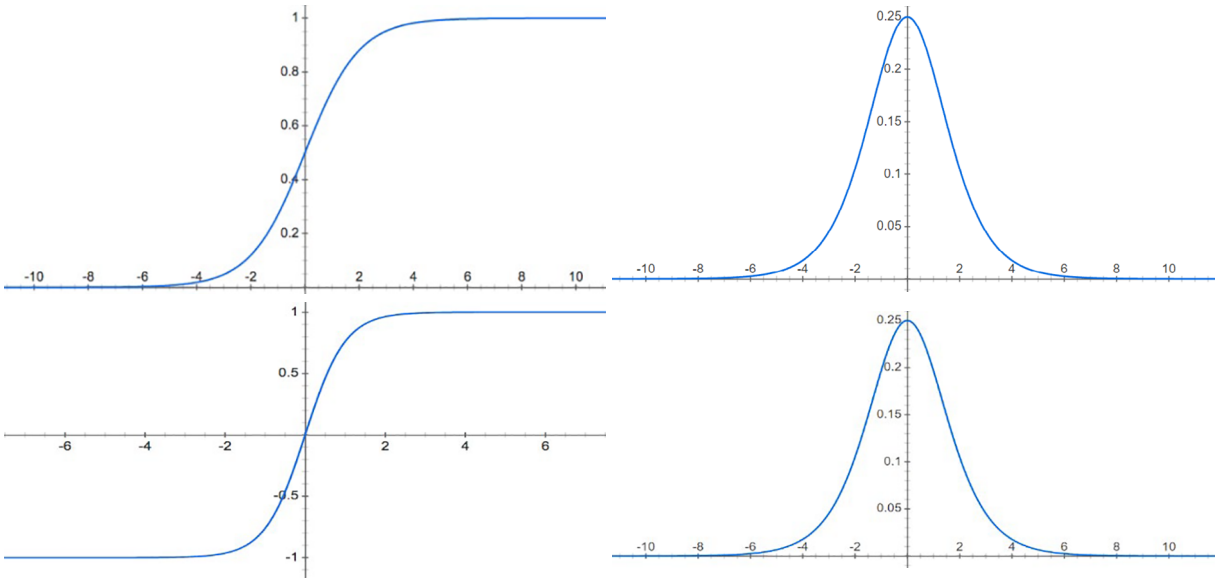
18. ábra: A dilatáció hatása: egy 3 x 3-as szűrő 1 dilatáció melett a sötétkék, míg 2 dilatáció esetén a piros színű pixelek értékeit használja

Forrás: saját készítés

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 (18. ábra) 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 (Goodfellow–Bengio–Courville 2016).

19. ábra: A pooling operátorok működése

Forrás: saját készítés



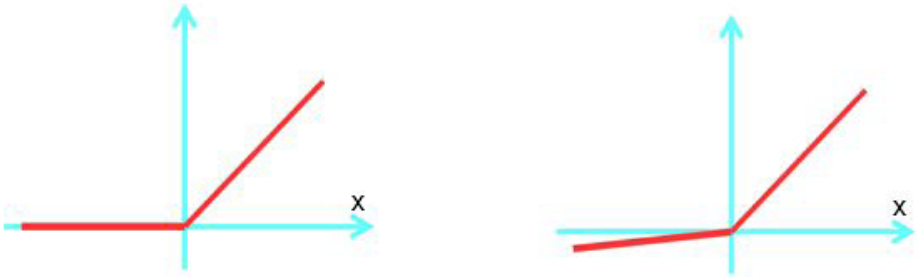
20. ábra: A szigmoid (felül) és a tanh (alul) aktivációk
és deriváltjaik a jobb oldalon
Forrás: saját készítés

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 (20. ábra) 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.

A jelenlegi hálók esetén a legnépszerűbb aktivációs függvény a ReLU (Rectified Linear Unit), 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 a ReLU használata esetén is előfordulhat az elülső rétegek beragadása, amelyre a paraméteres ReLU (PReLU), valamint a szivárgó (Leaky) ReLU (21. ábra) 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 (Goodfellow–Bengio–Courville 2016).

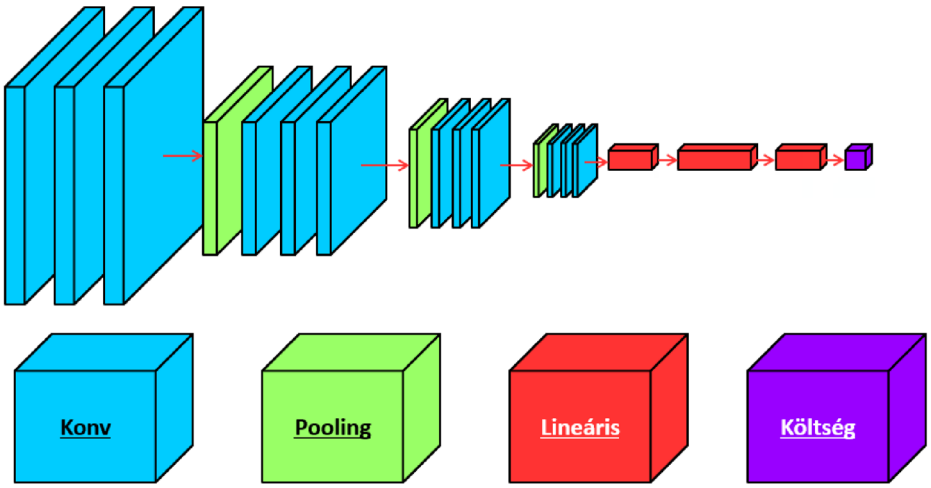
$$ReLU(x) = max(0, x) \qquad PReLU(x) = max(ax, x)$$

A jelen fejezetben bemutatott rétegeket tartalmazó neurális hálózatokat konvolúciós neurális hálózatoknak nevezzük, elsősorban a számítógépes látás területén alkalmazzák őket. 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) közbeékelésével. A háló végén tipikusan egy vagy több lineáris réteg állítja elő a végső kimenetet.



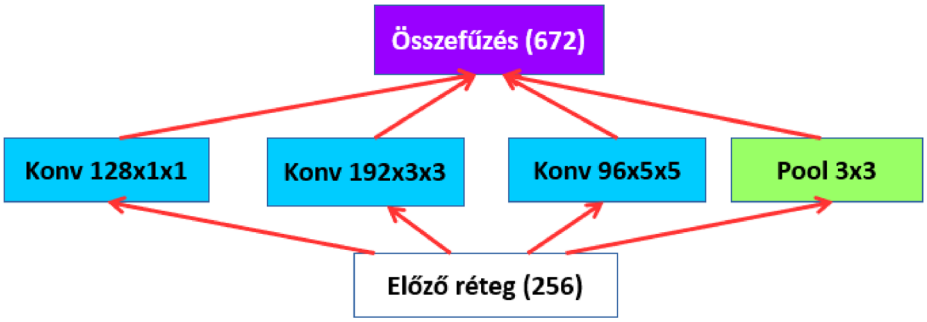
21. ábra: A ReLU (balra) és a PReLU (jobbra) aktivációk
Forrás: saját készítés

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 pixelkombinációk 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 pixeleknek, hanem ezeknek az alacsonyabb szintű jellemzőknek bizonyos kombinációira aktiválódnak. Belátható ez alapján, hogy egy sokrétegű konvolúciós neurális háló (22. ábra) 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 (Goodfellow–Bengio–Courville 2016).



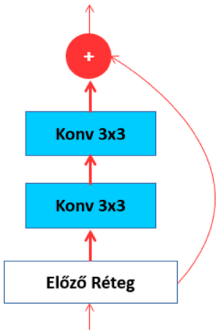
22. ábra: Tipikus konvolúciós neurális háló felépítése
Forrás: saját készítés

A fent ismertetett egyszerű architektúrának számos fejlettebb változata létezik. Ezen fejlesztéseknek általában két alapvető motivációja van: az egyik, hogy a mély neurális hálók numerikusan problémás konvergenciatulajdonságait valamilyen módon javítsuk. A másik, hogy olyan struktúrákat alkossunk, amelyek minél kevesebb szabad paraméter mellett minél komplexebb összefüggéseket tudnak megtanulni, ennek segítségével ugyanis a túlillesztés jelensége csökkenthető. Ez utóbbira jó példa az Inception (Szegedy et al. 2015) névre hallgató rétegtípus (23. ábra). Ennek a megoldásnak a lényege, hogy a konvolúciós rétegek nemcsak sorban, hanem egymással párhuzamosan is létezhetnek. Ezek a párhuzamos rétegek általában különböző méretű konvolúciókat hajtanak végre, egy olyan hálóstruktúrát eredményezve így, amely lényegesen jobban tudja kezelni az objektumok skálájában fellépő variációkat.



23. ábra: Az inception modul felépítése. A feltüntetett szűrőmélységek csak példák
Forrás: saját készítés

A konvergencia javítására az úgynevezett reziduális (HE–ZHANG–REN–SUN 2016) blokk jó példa. Ez a réteg (24. ábra) 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 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.



24. ábra: A reziduális blokk felépítése
Forrás: saját készítés

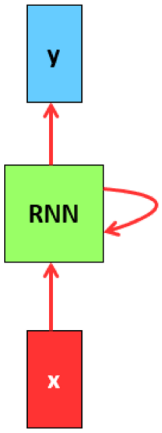
Érdemes még megemlíteni az úgynevezett tömörítő (bottleneck) rétegeket, amelyeket mind az Inception, mint a reziduális blokkok alkalmaznak. Ezek motivációja az, hogy a kétdimenziós konvolúciók elvégzése rendkívül drága, amennyiben az aktivációs térképek csatornáinak száma nagy. Éppen ezért ezekben a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

3.2.3. Visszacsatolt hálók

Ez eddigi diszkusszió során olyan módszereket ismertünk meg, amelyek állóképek egymástól független feldolgozására alkalmasak. Könnyen belátható, hogy az előrecsatolt konvolúciós hálóknak memóriaeleme nincs, így nem alkalmas időbeli sorozatok

feldolgozására. Érdemes megjegyezni, hogy amennyiben csak nagyon rövid időbeli összefüggéseket kell tudni kezelni, akkor lehetséges a háló bemenetére egyszerre két vagy több képet adni, vagy esetlegesen két kép különbségét felhasználni (Goodfellow–Bengio–Courville 2016).

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 a képek feliratozása, amelynek 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 a kimenete értelmezhető sorozatként.

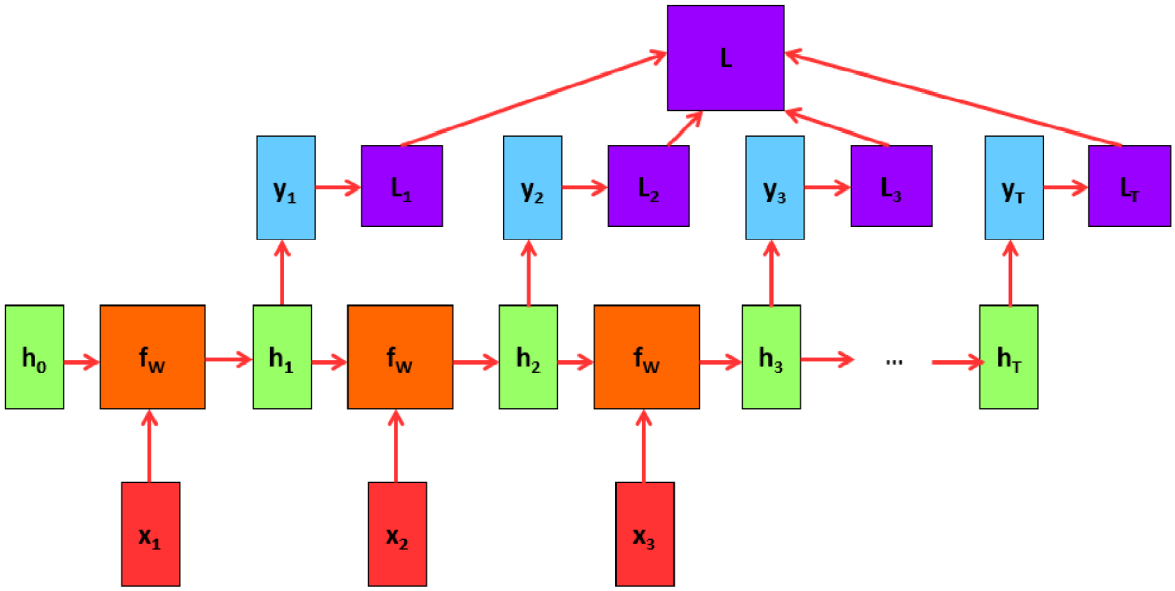


25. ábra: Egy tipikus visszacsatolt neurális háló architektúrája
Forrás: saját készítés

Sorozatok hatékony feldolgozásához viszont egy olyan új hálóstruktúrára 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 (25. ábra) egyenlete az alábbi módon adódik:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \qquad y_t = W_{hy}h_t$$

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 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 (26. ábra) műveletével. Ez azt jelenti, hogy az egyetlen RNN-réteg különböző időpontokban felvett állapotára úgy tekintünk, mint egy hagyományos háló egymást követő rétegeire (Goodfellow–Bengio–Courville 2016).



26. ábra: Egy RNN-háló időbeli kibontása

Forrás: saját készítés

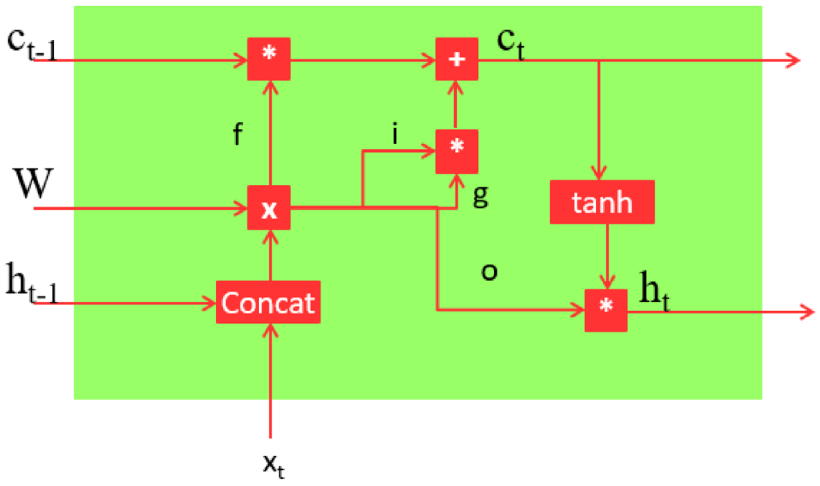
Mivel egy RNN-cellának minden időpontban van kimenete és hibája, ezért a kibontott háló minden rétegéhez fog tartozni egy-egy kimenet és hiba, amelyeknek összege adja ki a teljes hibát. Innentől a már megismert backpropagation algoritmus minden további nélkül használható. Két fontos különbség adódik 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.

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 ellehetetleníti a tanítást.

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 (Hochreiter–Schmidhuber 1997). A cella elnevezése onnan ered, hogy a készítői egy olyan rövid távú 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 LTSM négy aktivációs függvényt is tartalmazó egységből áll, amelyeket kapuknak nevezünk.

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

Az LTSM-cella (27. ábra) működése során az egyes kapuk hatása nélkül a *c* cellaállapot korábbi értéke változás nélkül átíródik az aktuális állapotba, így a kettő közötti derivált pontosan egy. A cellaállapot értéke azonban még az egyes kapuk hatására módosulhat. Az első ilyen kapu a felejtés kapu *f*, amely egy, a cella állapottal azonos méretű vektor, amelynek minden eleme nulla és egy között van a szigmoid nemlinearitás hatására. A cellaállapot vektorát ezzel a vektorral elemenként megszorozva a cellaállapot bizonyos részleteit elfelejtjük.



27. ábra: Az LSTM-cella felépítése
Forrás: saját készítés

A következő kapu az úgynevezett főkapu *g*, amelynek feladata, hogy a bemenet aktuális értékéből és a cellakimenet korábbi értékéből kinyerje azokat a jellemzőket, amelyek a cellaállapotban 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 cellaállapothoz hozzáadjuk. A végső lépés a cella aktuális kimenetének előállítása, amire a cellaállapotnak az *o* kimeneti kapu által szűrt értékeit adjuk ki.

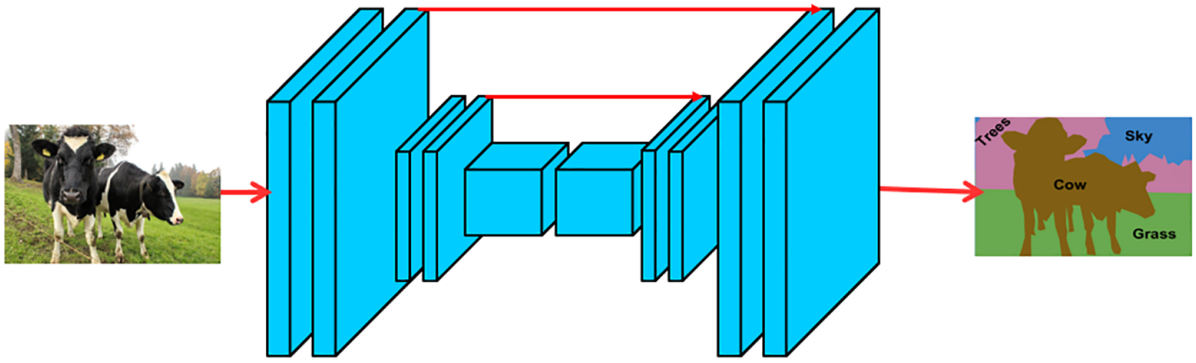
Érdemes megjegyezni, hogy az LTSM-cellának számos apróságokban eltérő variációja létezik, valamint léteznek nagyobb eltérést mutató, de hasonló alapötletre épülő visszacsatolt cellák. Ilyen például az úgynevezett kapuzott visszacsatolt cella (GRU – Gated Recurrent Unit). Szintén érdemes észrevenni, hogy 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.

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 egy 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.

3.2.4. Detektálás és szegmentálás

A fejezet bevezetésében a számítógépes látás több fontos feladatát is felsoroltuk, egyelőre azonban csak az osztályozás megvalósítását ismertettük. A jelenlegi fejezetben az objektumdetektálás és a szegmentáció különböző fajtáit vizsgáljuk. Az osztályozáshoz a legközelebb álló feladat a szemantikus szegmentálás, amelynek 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.

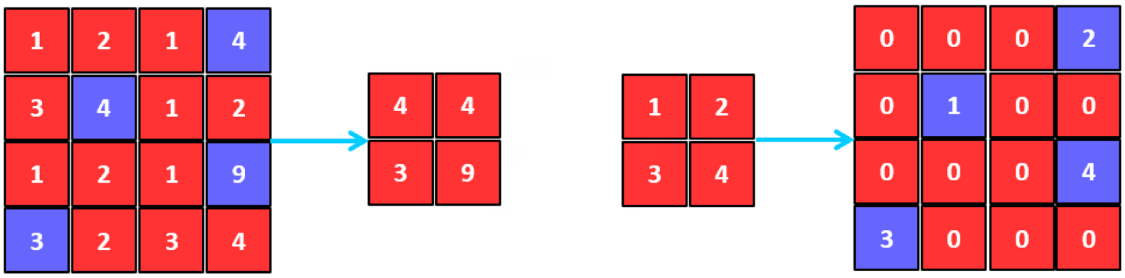
Éppen ezért célszerű lenne a folyamatot párhuzamosítani úgy, hogy egyetlen neurális háló segítségével elvégezhető legyen. Erre a teljesen konvolúciós hálózatok (FCN – Fully Convolutional Network) (Shelhamer–Long–Darrell 2017) alkalmasak, amelyek egyszerűen konvolúciós és aktivációs rétegek sorából állnak (28. ábra). 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.



28. ábra: Az FCN-háló architektúrája
Forrás: saját készítés

A gyakorlatban használt FCN-hálók egy fel- és leskálázó részből állnak, amelyek 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.

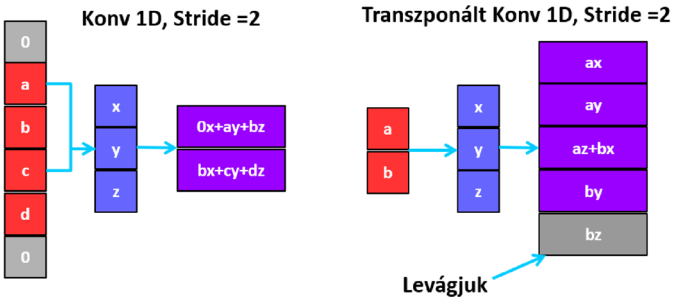
Az egyetlen megmaradó kérdés azonban az, hogy hogyan lehetséges a felskálázást konvolúciós neurális hálózatokban megvalósítani. Erre az egyik legegyszerűbb ötlet az úgynevezett unpooling operáció (29. ábra), melynek működése annyiból áll, hogy a leskálázó részben elvégzett maximum pooling során eltároljuk, hogy melyik pixelpozícióban volt a maximum érték, a felskálázás során pedig ebbe a pozícióba írjuk az alacsonyabb szint értékét, míg a többi pozíció nulla marad.



29. ábra: A max unpooling operáció működése
Forrás: saját készítés

Szintén elterjedt megoldás a transzponált konvolúció (30. ábra), amely tulajdonképpen a stride-dal végzett konvolúció megfordítása. A módszer elnevezése onnan ered, hogy a konvolúció leírható úgy, 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. Létezik még egy harmadik elterjedt módszer is, ez a sűrű felskálázó konvolúció (Wang et al. 2017). 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, csatornáinak száma pedig egyezzen meg az eredetivel. Ez a módszer szintén tanuló felskálázás, viszont több paraméterrel rendelkezik, így képes komplexebb transzformációk megtanulására lassabb működés árán.

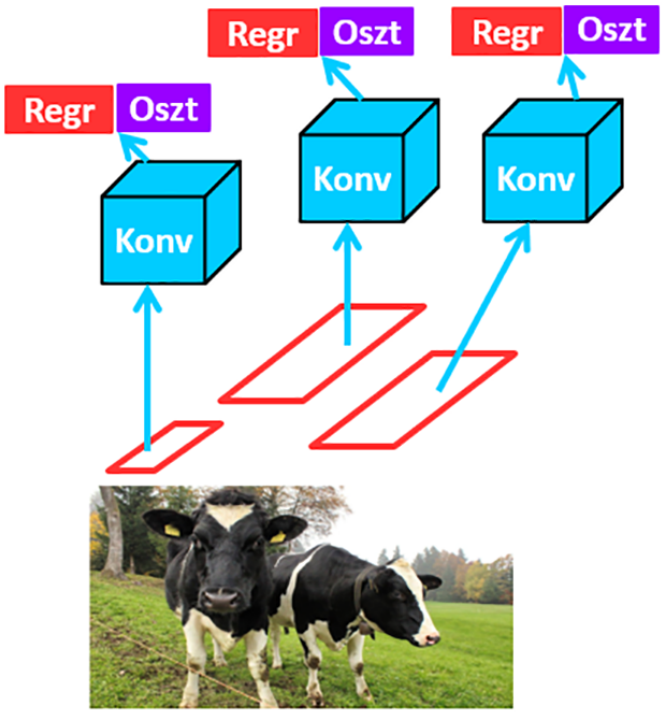
A következő fontos látási feladat a lokalizáció, amikor az osztályozás feladatát az adott osztályú objektum befoglaló téglalapjának meghatározásával egészítjük ki. Ez a feladat szintén könnyedén elvégezhető neurális hálók segítségével, hiszen nincs más dolgunk, mint egy osztályozó hálóhoz újabb négy 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.



30. ábra: A le- és felskálázó konvolúciók illusztrálása egydimenziós esetben
Forrás: saját készítés

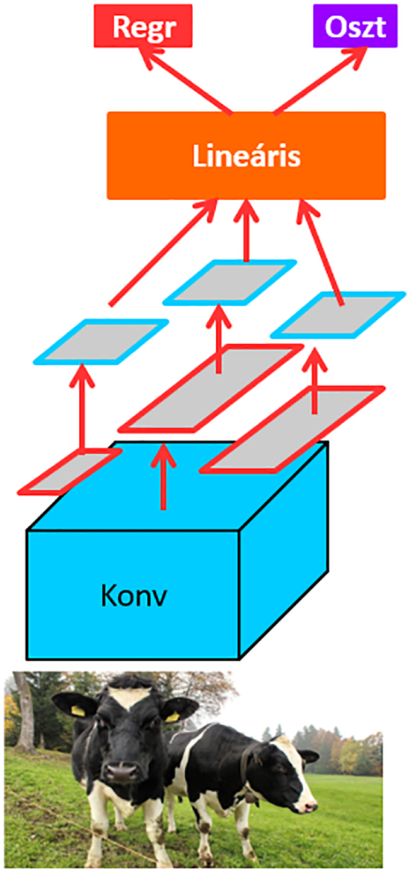
A detektálás feladata esetén azonban már lényegesen nehezebb dolgunk van. Ennek oka, hogy akárhány, akármilyen osztályú objektum előfordulhat, az architektúrát pedig ennek fényében kell megalkotnunk. Természetesen az objektumok számára egy durva felső becslést adhatunk, így készíthetnénk egy olyan konvolúciós hálót, aminek pontosan ennyi különböző osztályozó és téglalapbecslő 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.

Alternatív megoldást jelenthet, ha felhasználjuk a korábbi kötetben említett 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 (31. ábra). Ezt a módszert R-CNN (Girshick–Donahue–Darrell–Malik 2014) (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.



31. ábra: Az R-CNN-architektúra működése Forrás: saját készítés

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 (Girshick 2015). Az egész képen futtat egy csak konvolúciós és leskálázó rétegekből álló hálót, majd az ezáltal 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 régiójavaslatokon már csak egy kis méretű, csak lineáris rétegekből álló hálót futtat, amely az osztály és a téglalap becslését végzi (32. ábra). Ez a módszer az eredeti R-CNN-módszerhez képest 10-20-szor gyorsabban működik.



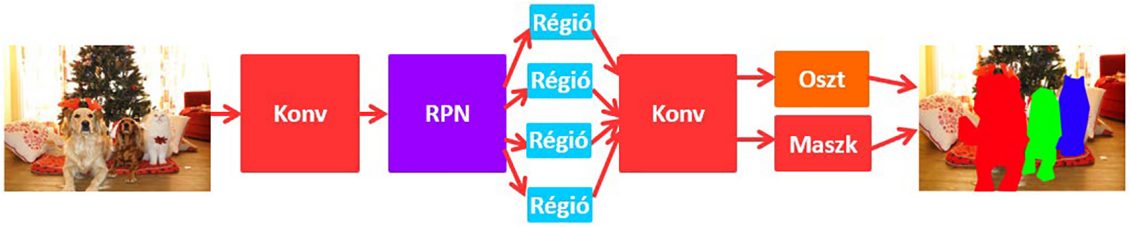
32. ábra: A FastR-CNN-architektúra
Forrás: saját készítés

A FastR-CNN működésének a leglassúbb 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) (Ren–He–Girshick–Sun 2017) nevű neurális háló segítségével végzi el. Ez a háló a kezdeti konvolúciós rész által előállított aktivációs térképből állít elő fix számú régiójavaslatot, amelyek mindegyikét binárisan osztályozza (objektum/nem objektum). Erre azért van szükség, mert a fix számú régiókimenet 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.

Fontos azonban tudni, hogy nem csak régiójavaslatok segítségével lehet hatékony objektumdetektálást végezni. Erre kitűnő példa a rendkívül népszerű YOLO (Redmon–Divvala–Girshick–Farhadi 2016) (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 N x N-es rács segítségével felosztja, és minden rácsból B darab objektumjelölt téglalapot becsül. Minden téglalaphoz tartozik egy bináris osztályozó is, amely az adott téglalapba eső képrészlet „objektumszerűségét” adja meg. Az architektúra fő újítása azonban, hogy nem tartozik minden téglalaphoz külön osztályozó kimenet, hanem minden azonos rácsponthoz tartozó téglalap ugyana-

zon az osztályozó kimeneten osztozik. Mivel az osztályozó kimenetek általában nagyok, ezért ezzel az ötlettel a YOLO-architektúra a kimenetek számát nagymértékben tudja csökkenteni. Ennek a megoldásnak nyilvánvaló hátránya, hogy az egymáshoz közel lévő, eltérő osztályú objektumok esetén a háló nem tud teljesen helyes döntést hozni. A hátrány ellenére a YOLO az egyetlen olyan objektumdetektálási architektúra, amely közepesen erős hardveren is képes valós időben futni.



33. ábra: A mask R-CNN működése
Forrás: saját készítés

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 (33. ábra). Ezt könnyedén megtehetjük a szemantikus szegmentálásból ismert felskálázó hálórész segítségével. Ezt az architektúrát Maszk R-CNN (He–Gkioxari–Dollár–Girshick 2017) néven ismerik.

3.2.5. Tanítás a gyakorlatban

A korábbi fejezetek során megismertük a mély tanulás és a konvolúciós neurális hálók alapjait, valamint részletesen tárgyaltuk 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, amelyek a módszerek használatát nehézzé teszik. A jelen alfejezet célja, hogy összeszedje azokat a gyakorlati megfontolásokat és praktikákat, amelyek nélkül rendkívül nehéz a való életben jól működő neurális hálókat létrehozni (Goodfellow–Bengio–Courville 2016).

A neurális hálók tanítását alapvetően három dolog nehezíti meg:

1. A túlillesztés (overfitting) jelensége, amely a betanított modell való életben történő alkalmazhatóságát veszélyezteti.
2. Numerikus problémák, amelyek az optimalizáló algoritmus konvergenciáját hiúsítják meg.
3. A hálók tanításához szükséges nagy mennyiségű címkézett adathalmaz előállításának problémája.

A mély tanulásról szóló első alfejezetben bevezettük a gradiens módszert, amely a hibafüggvény deriváltjának segítségével iteratívan módosította a háló paramétereit a teljesítmény javításának érdekében. Arról viszont szándékosan hallgattunk, hogy hogyan kapjuk meg a kezdeti paraméterértékeket. A helyzet az, hogy a problémát helyesen megoldó paraméterekről a 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, amelynek 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, amelynek következtében a háló gradiensei is, így a háló a kezdeti értékekbe beragad.

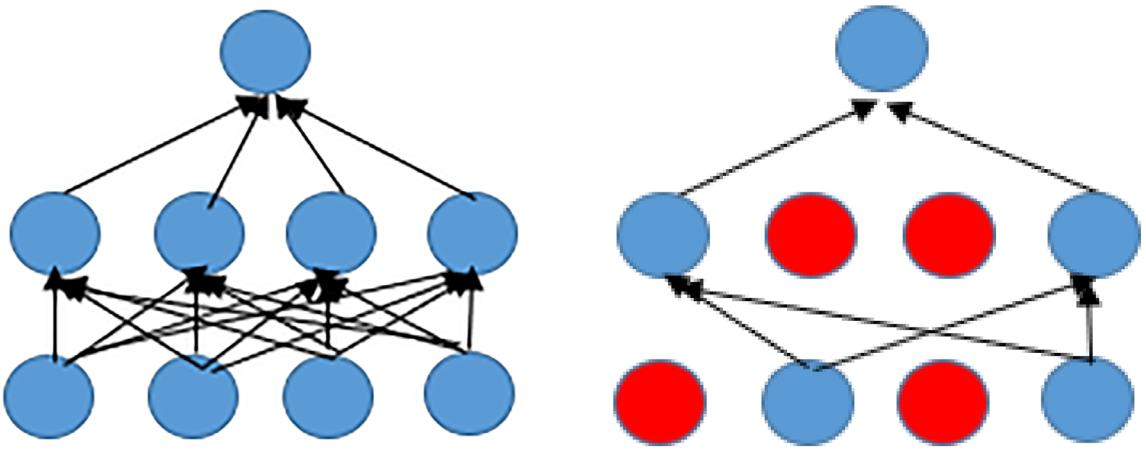
Ennek elkerülésére a háló súlyainak szórását nagy körültekintéssel kell megválasztani, hogy se túl kicsik, se túl nagyok ne legyenek. Ennek több módja is létezik, melyek közül a leginkább elterjedt választások a Xavier, illetve a He (He–Zhang–Ren–Sun 2015) inicializációs formulák, amelyek a következőképp határozzák meg a háló egyes rétegeinek kezdeti súlyainak szórását:

$$var_X(W) = \frac{2}{n_i + n_o} \qquad var_{He}(W) = \frac{2}{n_i}$$

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.

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 pixelértékeinek eloszlása megközelítőleg sztenderd normális legyen. Hiába inicializáljuk ugyanis jól a háló súlyait, ha a bemenet értékei 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 numerikus problémák léphetnek fel.

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 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.



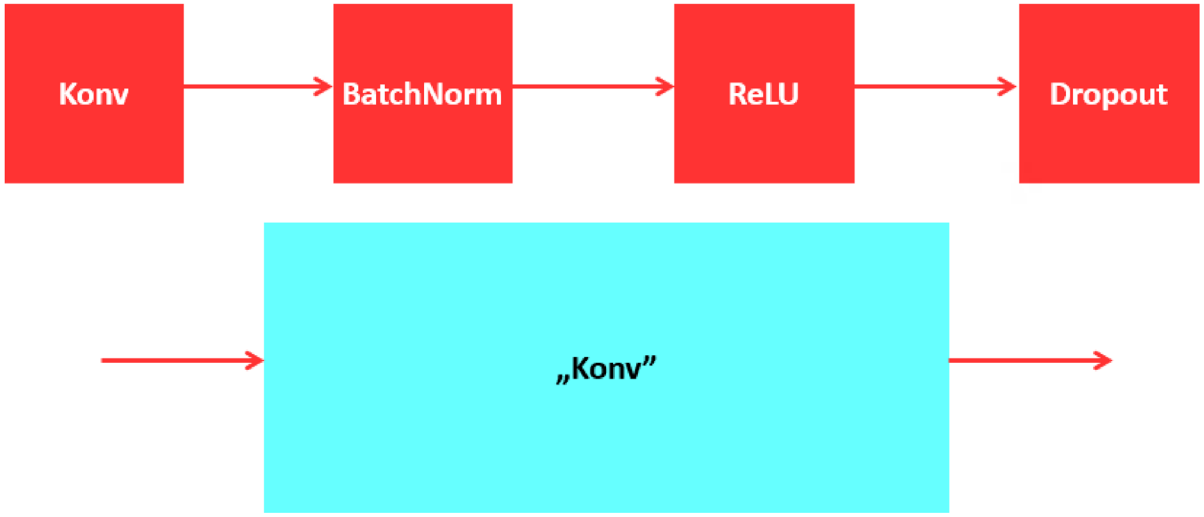
34. ábra: A dropout alkalmazása
Forrás: saját készítés

A nemkívánt aktivációk elkerülésére két további, gyakran alkalmazott módszer létezik. Ezek közül az első a dropout (Srivastava–Hinton–Krizhevsky–Sutskever–Salakhutdinov 2014) 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 (34. 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.

A másik megoldás az úgynevezett batch normalizálás (Ioffe–Szegedy 2015), 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 minibatchnek megfelelő (általában 32 többszöröse) képet értékelünk ki. A batchnormalizálás 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 nemcsak a túlillesztést mérsékeli az aktivációk eloszlásának normalizálásával, hanem a numerikus konvergenciát is javítja. A batchnormalizálás képlete az alábbi:

$$x_{out} = \alpha \frac{(x - \mu)}{\sigma} + \beta$$

Ahol μ és σ az *x* bemenetek becsült átlaga és szórása, míg és tanult paraméterek. Érdemes megjegyezni, hogy modern neurális hálókban a batchnormalizálás teljesen alapvető művelet, így általában minden konvolúciós réteget követ egy ilyen réteg. A batchnormalizálás és a dropout akár együttesen is használható (35. ábra), bár a legtöbb kísérlet minimális teljesítménynövekedést mutat csak.



35. ábra: Egy általános konvolúciós réteg valójában ezen rétegek egymás után történő építéséből valósul meg. Batchnormalizálás esetén a dropout alkalmazása opcionális

Forrás: saját készítés

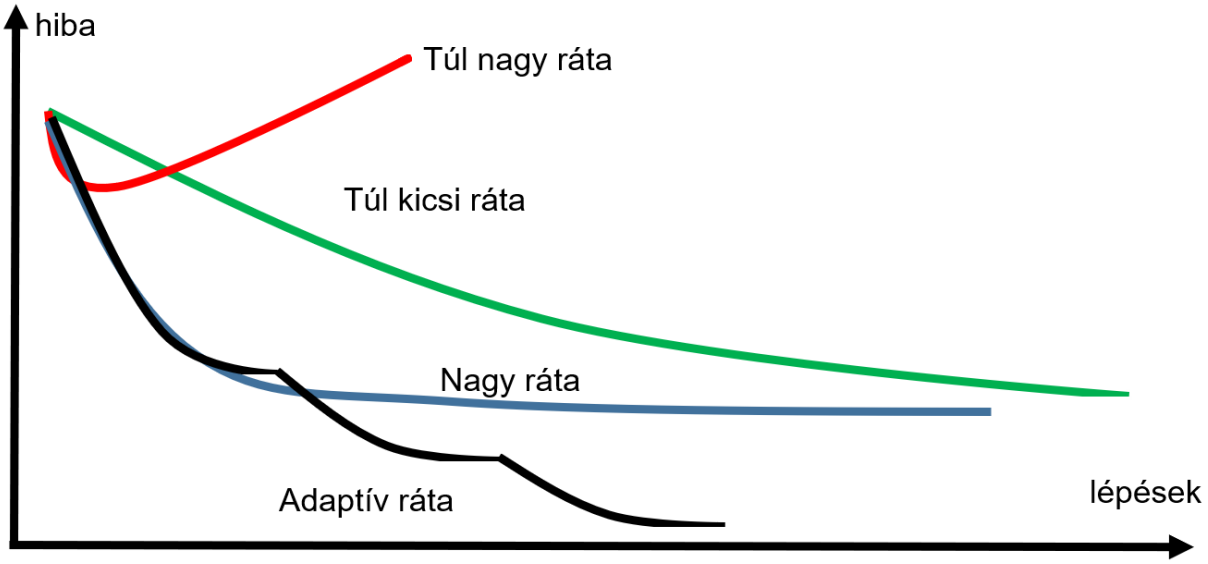
A túlillesztés jelenségének van még egy lehetséges elkerülési módja: gondoljunk bele, hogy a túlillesztés esetén a neurális háló a tanító adatbázis elemeire egyesével tanulja meg a helyes választ. Nyilvánvaló, hogy minél több tanító adat áll rendelkezésre, annál nehezebb ezt megtenni. Éppen ezért a tanító 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 adataugmentációra (Goodfellow–Bengio–Courville 2016). 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 batchnormalizálás, regularizáció és adataugmentáció módszereit együtt használjuk.

A neurális hálók tanításának további nehézsége, hogy egy tipikus tanítás során több tucat hiperparaméter is rendelkezésünkre állhat, amelyek 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.

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 (Goodfellow–Bengio–Courville 2016).

A gyakorlatban ezen megfontolások miatt nem egyetlen tanulási rátát szokás alkalmazni. Ehelyett a tanítás kezdetén a legnagyobb olyan tanulási rátával indítjuk az optimalizálást, amivel a hiba értéke stabil csökkenést mutat, így a lehető legyorsabban 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, amelyek közül az egyik leggyakoribb a ráta fix faktorral történő csökkentése bizonyos számú lépés után (36. ábra).



36. ábra: Különböző tanításiráta-választások hatása a tanítás konvergenciájára
Forrás: saját készítés

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 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 koszinuszfüggvény első fél periódusának megfelelően állítja. A koszinuszlágyítás 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 (Loshchilov–Hutter 2017).

A neurális hálók tanításának harmadik nehézsége a nagyszámú címkézett tanítóadat előállítása. 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 újratanítani, amihez lényegesen kevesebb adat elegendő, hiszen kevesebb szabad paramétert tartalmaznak, mint az egész háló.

Ezt a technikát transzfertanulásnak nevezzük, és elterjedt megoldás a mély tanulás területén. A fent ismertetett érvelés annyira igaz, hogy számos esetben elegendő a hálók legutolsó, lineáris rétegét újratanítani. Más esetekben szükség lehet az elülső hálórészek finomhangolására, azonban ehhez is nagyságrendekkel kevesebb adat elegendő lehet. Egy további technika a hálók teljesítményének növelésére a modellegyüttesek használata. Ekkor több, függetlenül tanított (és gyakran eltérő architektúrájú) neurális háló eredményeit átlagoljuk össze, ami a pontosságot akár 2-3%-kal is növelheti. Az együttesek használata a korábban említett ellenséges példák ellen is nyújthat valamekkora védelmet (Goodfellow–Bengio–Courville 2016).

A jelen fejezet utolsó témája a neurális hálók gyakorlati felhasználásra való felkészítésének (installálásának) kérdései. A gépi tanulás alapjainál említettük, hogy a tanításhoz két külön adathalmazt érdemes használunk, a gyakorlatban azonban ez 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 tesztadatbázist érdemes használni. A módszer megbízhatóságához elengedhetetlen, hogy ezt a három adatbázist teljesen elkülönítsük, és csak egyetlen célra használjuk.

Neurális hálózatok telepítése esetén két probléma adódik: A neurális hálók ugyanis többmillió 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. Szerencsére léteznek technikák neurális hálók gyorsítására. Ezek közül az egyik legfontosabb a tisztítás művelete (pruning) (Huang–Zhou–You–Neumann 2018), amely során a háló súlyai közül kiválasztjuk a legkevésbé fontos néhány százalékot, és ezeket töröljük. A súlyok rangsorolására számos módszer létezik, amelyek közül a legegyszerűbb 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%-ás törölni lehet csupán néhány százalékos teljesítménybeli veszteség mellett. Ez nyilvánvalóan tízszeres gyorsulást eredményez.

Hasonlóan hatékony módszer a súlyok kvantálása, amelynek 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.

Fontos még megemlíteni, hogy a mély tanulás algoritmusai meglehetősen nagy számítási kapacitást igényelnek, így használatukhoz érdemes speciális hardveregységeket alkalmazni. Mivel a mély konvolúciós neurális hálók működésének nagy része kifejezhető mátrix- és tömbműveletekkel, ezért rendkívül jól párhuzamosíthatók. Mind a neurális hálók tanítására és futtatására célszerű grafikus feldolgozó egységeket (GPU) használni. Az utóbbi időben megjelentek külön a mátrixműveletek elvégzésére specializálódott feldolgozó egységek (TPU – Tensor Processing Unit) is GPU-k részeként, valamint különálló hardver formájában is.

3.3. Nem felügyelt tanulás

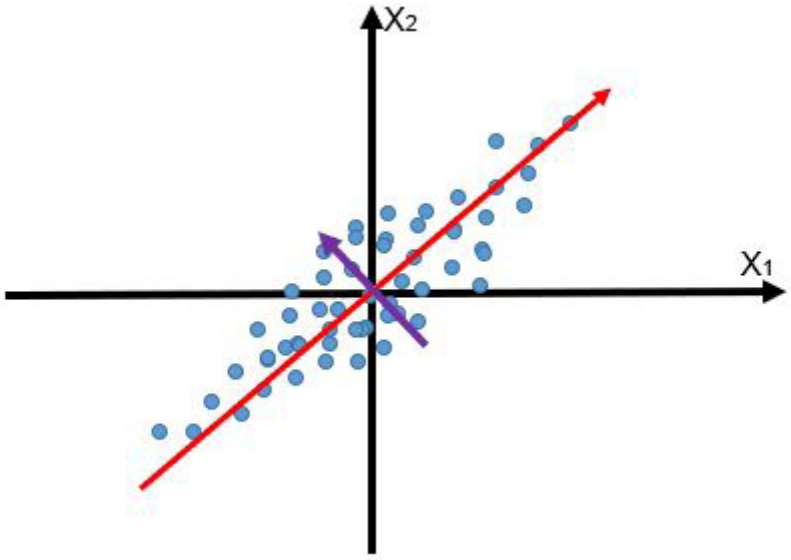
A fejezet eddigi részében a felügyelt tanulás módszereit ismertettük. Bár a tanulásnak ez a módszere rendkívül hasznos, mégis jelentős korlátai vannak. Egyrészt a felcímkézett tanító adatbázisok előállítása rendkívül hosszadalmas és drága feladat. Másrészt a címkézés minősége alapvetően korlátozza a 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.

Erre a problémára igyekeznek megoldást adni a nem felügyelt és a megerősítéses tanuló algoritmusok. Az előbbi algoritmuscsalád esetében a tanító adatbázisban csak lehetséges bemenetek találhatók, így ilyen adatbázisokat rendkívül olcsó előállítani, mivel a legtöbb esetben az új adatok beszerzése automatizálható. Az algoritmusok célja, hogy a bemeneten látott adatokat valamilyen kompakt modell segítségével magyarázzák, azoknak a belső struktúráját feltérképezzék. Ilyen algoritmusokra jó példák a korábbi kötetben ismertetett klaszterezési eljárások (k-Means, MoG), amelyek voltaképpen az osztályozás felügyelet nélküli változatának tekinthetők. Az előző alfejezet elején röviden ismertetett TLS-módszer is értelmezhető a lineáris regresszió felügyelet nélküli változataként.

Egy másik gyakori problémája a felügyelet nélküli tanulásnak a dimenzióredukció feladata. Ez a probléma abban az esetben áll elő, ha egy olyan bemeneti adathalmaz áll rendelkezésre, amelyben a változók dimenziószáma meglehetősen 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.

A dimenzióredukció egyik legalapvetőbb módszere a főkomponens-analízis (James–Witten–Hastie–Tibshirani 2009) (PCA – Principal Component Analysis). 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 koordináta-rendszerbe transzformálja, amelynek a bázisvektorait az adathalmaz kovarianciamátrixának sajátértékei adják. Az így megkapott bázisvektorokat főkomponensnek nevezzük, és az ezek segítségével definiált új változók már statisztikailag függetlenek lesznek (37. ábra).

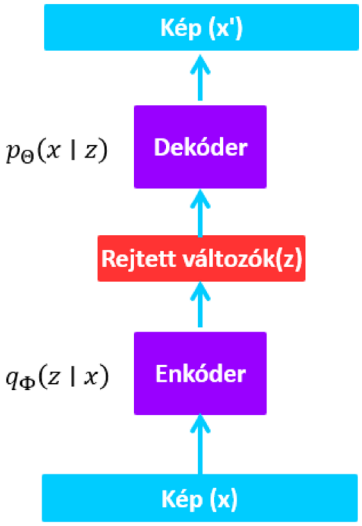
Ezt követően az így megkapott változók közül a legkisebb varianciájúakat eldobjuk egészen addig, amíg az adatbázis teljes varianciájának bizonyos százaléka alá nem estünk. Ez egyes főkomponensek varianciáját a sajátvektorhoz tartozó sajátértékek adják meg. Érdemes megjegyezni, hogy a főkomponens-analízis 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.



37. ábra: A főkomponens-analízis működési elve. Piros színnel jelöltük a legnagyobb, lila színnel pedig a legkisebb szóródás irányát
Forrás: saját készítés

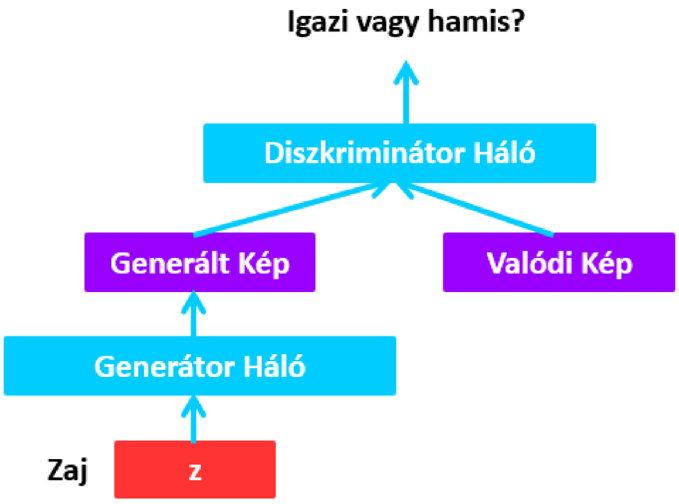
A főkomponens-analízis egy érdekes alkalmazása az eigenfaces, vagyis sajátarcok arcfelismerő rendszer, ahol egy emberi arcokból álló adatbázisból választották ki a legfontosabb sajátvektorokat. Ezek a sajátvektorok egyrész felhasználhatók az emberi arcok különböző variációinak megértésére, valamint osztályozására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

Érdemes még megjegyezni, hogy a főkomponens-analízisnek létezik egy felügyelt tanuláshoz használható változata is, a lineáris diszkriminancia-analízis (LDA) (James–Witten–Hastie–Tibshirani 2009). 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 az LDA nem a teljes adathalmaz varianciáját, hanem az osztályok közötti varianciát igyekszik maximalizálni.



38. ábra: A variációs autoenkóder architektúrája
Forrás: saját készítés

Fontos még megemlíteni, hogy az előző alfejezetben részletesen tárgyalt neurális hálók használhatók felügyelet nélküli tanuló algoritmusként is. Ezeket a hálózatokat általában generatív neurális hálóknak nevezzük, ugyanis képesek képek vagy videók generálására. Ezen hálózatok közül az egyik legfontosabb az úgynevezett variációs autoenkóder (VAE) (KINGMAN–WELLING 2013). Ez az algoritmus egy fel- és leskálázó részből áll (38. ábra), amelyet arra tanítanak, hogy a bemenetére adott képet legyen képes minél pontosabban visszaállítani. Az autoenkóder felskálázó része felhasználható képek generálására, ha a bemenetére megfelelő eloszlású véletlen zajt adunk.



A generatív neurális hálók legfejlettebb módszere az úgynevezett GAN (GOODFELLOW ET AL. 2014) (Generative Adversarial Network) háló. A GAN valójában két külön neurális hálóból áll, az egyik egy bináris osztályozó, míg a másik egy felskálázó jellegű generátorhálózat (39. ábra). A GAN tanítása során a generátorháló véletlen zajból egy képet generál, majd a másik háló (a diszkriminátor) megpróbálja eldönteni, hogy a kép generált vagy valódi. A diszkriminátorháló tanításához valódi képeket is használunk. Bár a valósághű képek generálásának felhasználhatósága meglehetősen limitált, az itt rövidesen ismertetett módszerek relevanciája abban rejlik, hogy címkézett adatok felhasználása nélkül képesek a képek felépítését megérteni. Az így kapott hálózatok jól használhatók transzfer tanulásra.

4. KOMPLEX LÁTÓRENDSZEREK

A korábbi fejezetek során részletesen bemutattuk a számítógépes látórendszerek két jelentős területét. A jelen fejezet fókuszában azok a közigazgatási szempontból releváns megoldások lesznek, amelyek látórendszereket alkalmaznak. A fejezet első részében a köz- és közlekedésbiztonságot támogató megfigyelési rendszereket fogjuk bemutatni, külön kitérve az egyes események és azok szereplőinek azonosításának kérdéseire. Ezt követően ismertetjük az okmányok és dokumentumok automatikus feldolgozásának, valamint hitelesítésének kérdéseit. Ezt követően kitérünk a különböző virtuális és kiterjesztettvalóság-rendszerekre, amelyeknek keretében a saját kutatásunkat is bemutatjuk. A fejezet utolsó részében egyéb, az előző kategóriákba nem sorolható rendszerekkel foglalkozunk, mint például az automatikus környezetértékelés.

4.1. Közbiztonság és megfigyelés

A köz- és közlekedésbiztonság fenntartása az állam egyik legfontosabb feladata, amelynek egyik alapvető eleme a védelem alá helyezett területek (legyen itt szó közterületekről, utakról, vagy akár épületekről) megfigyelése. Az utóbbi években egyre elterjedtebb a különböző kamerák kihelyezése, ezek felvételei azonban utólag nem minden esetben szolgálnak a hatóságok segítségére. A kamerák valós idejű monitorozása viszont hatalmas emberi erőforrásmennyiséget igényelne, ezért célszerű valamilyen intelligens látásra alapuló figyelmeztető rendszert használni a kamerákban, amely bizonyos eseményekre az egyes hatóságokat figyelmeztetni képes.

Automatikus számítógépes látásra épülő riasztórendszerek legegyszerűbb változata a mozgásérzékelésre alapuló detektálás. Ez a módszer főképp zárt, illetéktelen hozzáféréstől jól védett helyeken alkalmazható könnyedén. Ezek a módszerek gyakran meglehetősen egyszerű módszereken alapszanak, mint például ez egymást követő képkockák különbségén, majd ennek a különbségnek a küszöbölésén alapuló technikán. Ennél valamelyest bonyolultabb megoldás a monográfia előző kötetében ismertetett gauss háttérmodellre épülő mozgásdetektálás. Érdemes megjegyezni, hogy a megfigyelésre alkalmazott kamerák felvételeit szinte minden esetben rögzíteni szokták, ami számos kamera esetén hatalmas tárhelyet igényel. Azonban ha csak akkor rögzítjük az egyes kamerák felvételeit, amikor azok mozgást érzékelnek, akkor egy-két nagyságrenddel csökkentjük a szükséges tárhely méretét, mégis minden releváns információt rögzítünk (Upasana–Manisha–Mohini–Pradnya 2015).

Fontos azonban megjegyezni, hogy a mozgás tényének detektálásán felül rendkívül fontos lehet az egyes mozgó objektumok követése. Amennyiben ezt el tudjuk végezni, akkor információt nyerhetünk a mozgó objektum haladásának irányáról, valamint arról, hogy az esetleges alternatív útvonalak közül melyiken haladt tovább. Összetett, sokkamerás megfigyelési rendszerek esetében ez felhasználható arra, hogy megjósoljuk, hogy egy bizonyos mozgó objektum melyik kamerákban jelenhet meg legközelebb. Ennek egyik legegyszerűbb módja, ha az egyes mozgó objektumok széleit valamilyen egyszerű követési algoritmus (például) optikai áramlás segítségével követjük (Hossen–Tuli 2016), a mozgás irányát pedig az áramlásvektorok segítségével állítjuk elő. Érde-

mes még megjegyezni, hogy lokális jellemzők (például SIFT) felhasználhatók az egyes mozgó objektumok robusztus azonosítására, ha azok valamilyen oknál fogva nem a várt kamerákban jelennének meg.

A mozgásdetektáláson alapuló eseménydetektálást azonban csak akkor célszerű alkalmazni, ha a mozgás megléte valóban eseményértékű. Tipikusan a közterületek és közutak esetében a mozgás az alapértelmezett állapot, így önmagában semmilyen használható információt nem tartalmaz. Ilyen esetekben komplex eseménydetektáló eljárásokra lesz szükségünk, amelyek bizonyos általunk előre meghatározott eseménykategóriákat, vagy egyes esetekben a szokásos mintáktól jelentősen eltérő eseteket (úgynevezett anomáliákat) képesek észlelni. Könnyű azonban belátni, hogy komplex események észlelése már a korábban ismertetett jelenetértelmezés feladatkörébe esik, így a legtöbb ilyen rendszer a gépi tanulás vagy a mély tanulás eszköztárát alkalmazza. Az ilyen rendszereket gyakorta komplex eseményfeldolgozó (CEP – Complex Event Processing) rendszereknek nevezzük.

A CEP-rendszerek kutatása csak az utóbbi néhány évben kezdett el komolyabban foglalkozni a gépi tanulás módszereinek alkalmazásával, ezelőtt gyakorta egyszerű szenzorok (mozgásdetektálók, ajtónyitás-érzékelők stb.) jeleire támaszkodtak. Ezek azonban a fent leírtak miatt limitáltan alkalmazhatók. Az elmúlt néhány évben azonban egyre inkább előtérbe került a különböző gépi tanulásra épülő látórendszerek alkalmazása, amelyre jó példa Shabad et. al. (Shahad–Bein–Saad–Hussain 2016) által fejlesztett rendszer. Ez a megoldás egyszerű osztályozókat alkalmaz négy különböző esemény detektálására: egy adott helységből és a teljes épületből való ki- és belépés. Könnyen belátható, hogy ezek az események az egyszerű mozgásnál lényegesen magasabb szintűek, és habár az alkalmazás fókusza még mindig egyes épületek biztonsága, a módszer közterületeken is felhasználható egyes események detektálására.

Kifejezetten közlekedési jellegű alkalmazás Cui et. al. (Cui–Li–Chen–Li 2011) megoldása, akik abnormális közúti események detektálását végzik el lokális képjellemzők felhasználásával. Ez a rendszer is az úton haladó járművek mozgásalapú szegmentációját használja arra, hogy a feldolgozás szempontjából érdekes képrészletet kijelölje. Ezt követően a különálló objektumokat négy kategóriába sorolják a lokális képjellemzők segítségével elvégzett osztályozás alapján: gyalogos, jármű, összeolvadt járművek (ez a kamera nézőpontja miatt gyakran előfordul) és zaj. Miután az egyes járművek és gyalogosok mozgásának irányát és sebességét meghatározták, ezek segítségével számos abnormális esemény detektálása lehetséges: baleset, dugó, gyorshajtás, gyalogosok közé hajtó jármű, illetve egyéb közlekedési kihágások.

Ennél érdekesebb azonban Sultani et. al. (Sultani–Chen–Shah 2018) kutatása, akik munkájukban egy mélytanulás-alapú anomáliadetektálás-módszert javasolnak. Az anomáliák detektálásának egyik nehézsége, hogy az anomália nehezen definiálható, így a videók címkézése nem egyértelmű, továbbá az anomáliák meglehetősen ritkák, így a tanító példák nagy része nem az. Egy ilyen rosszul kiegyensúlyozott tanító adatbázis numerikus problémákat okozhat a tanítás során. Az általuk kifejlesztett módszer az anomáliák detektálását regressziós problémaként fogalmazza meg, ahol a cél, hogy az anomáliákat tartalmazó videók minél nagyobb anomália értéket kapjanak a tanuló algoritmustól. Munkájuk egyik másik fontos hozzájárulása, hogy készítettek egy valódi megfigyelési videókból álló adatbázist, amely 1900 videóból áll, amelyek 13 különböző anomáliakategóriát tartalmaznak. Ezek a kategóriák közlekedési és közéleti (lopás, betörés, vandalizmus, gyújtogatás stb.) jellegű anomáliákat is tartalmaznak.

Az események automatikus detektálása rendkívül hasznos funkció, hiszen lehetővé teszi az megfelelő hatóságok azonnali riasztását, valamint az egyes riasztások ember

által történő felülvizsgálata is lehetséges. Hasonlóan fontos azonban az egyes anomáliákban részt vevő szereplők azonosítása, ugyanis az egyes esetek helyszíni rendezése nem minden esetben lehetséges. Közúti esetekben a legtöbbször lehetőség van a járművek rendszámának automatikus meghatározására, amelyre számos hagyományos és mély tanulást alkalmazó módszer létezik (Ibrahim et al. 2013). Egyéb esetekben személyek arcalapú azonosítására kell szorítkozni, amely lényegesen nehezebb feladat. Személyek automatikus azonosítása és lokalizációja ugyan érdekes személyiségi jogi kérdés, a jelen kötetben a technológiai megvalósítás tárgyalására fogunk szorítkozni.

Automatikus arcfelismerés végrehajtásához szükség van egy adatbázisra, amelyben a felismerendő személyek arcairól referenciaképek találhatók. Az arcfelismerő módszerek ezt követően ezeket a referenciaképeket hasonlítják össze a videóban talált arcokkal. Ez a probléma meglehetősen nehéz egyrészt az arcok különböző természetes torzulásai (szemüveg, arcszőrzet, frizura és arckifejezés), valamint a megfigyelő videókban elérhető meglehetősen rossz felbontás miatt. Grace és Reshmi (Grace–Reshmi 2015) munkája a korábban ismertetett főkomponens-analízis (PCA) módszert alkalmazza arcok különböző jellemzőinek a képből történő kinyerésére (lásd: Eigenfaces). Ezt követően a PCA során megkapott sajátvektorok terében hasonlítja össze az éppen észlelt arcot a referencia-adatbázisban található mintákkal.

Arcfelismerés megvalósítására természetesen léteznek mély tanulást alkalmazó módszerek (Bashbaghi–Granger–Sabourin–Parchami 2018). Itt a kutatók célja, hogy a mély konvolúciós neurális hálók olyan képjellemzők előállítását tanulják meg, amelyek általánosan alkalmazhatók különböző személyek arcának megkülönböztetésére. Ehhez a tanítás során ugyanazon a hálón egyidejűleg három különböző bemenetet küldenek előre: egy referencia-arcképet, valamint egy, az adott arcot tartalmazó pozitív és az arcot nem tartalmazó negatív videót. A három különböző bemenetből előállított kimeneten pedig az úgynevezett triplet (trió) hibafüggvényt értékelik ki, és a tanításhoz ezt használják. A triplet hibafüggvény azt írja elő, hogy a háló kimenete legyen hasonló a referenciakép és a pozitív videó esetében, de a negatív videó kimenete legyen jelentősen eltérő.

4.2. Okmányok kezelése

A számítógépes látás egy másik jelentős közigazgatási felhasználása a különböző okmányok automatikus kezelése és feldolgozása, valamint hitelesítése. Ennek a feladatnak a legtöbb esete lényegesen egyszerűbb, mint a köztéri megfigyelés, hiszen az okmányok alapvetően mesterségesen előállított tárgyak, így a megjelenésük meglehetősen szabályos. Ettől függetlenül azonban a környezetük, vagyis a kép háttere még lehet valamilyen természetes tér, így a konkrét okmány háttértől való elválasztása bizonyos esetekben lehet problémás. Egyes okmányok eredetiségének vizsgálata kizárólag látás segítségével szintén problémás lehet, hiszen bizonyos eredetiséget igazoló markerek ellenőrzéséhez az okmány megfelelő mozgatása szükséges. Ebből kifolyólag ezt általában más módszer segítségével szokás elvégezni.

Személyi azonosságot igazoló okmányok automatikus olvasására már a kétezres évek elején is léteztek rendszerek, amelyek egyszerű szkennerberendezésen alapultak (Lladbs–Lumbreras–Chapaprieta–Queralt 2001). A rendszer alapvetően öt lépésből áll, amelyek közül első a képek készítése, melyet az előfeldolgozás és a szöveges részek detektálása követ. Ennek a lépésnek a lényege, hogy a képi hibáktól mentesítsük a képet, valamint hogy a dokumentumot befoglaló téglalapot meghatározhassuk. Ezt követően a dokumentum mérete és színtulajdonságai alapján a rendszer az okmány típusáról döntést hoz. Az okmány típusának ismeretében már megtalálhatjuk rajta azo-

kat a releváns régiókat, ahol az elolvasandó szövegrészek találhatók. A rendszer utolsó lépése az optikai karakterfelismerés, amely nyomtatott karakterek esetén már ekkor is meglehetősen nagy pontossággal működött.

Az azóta eltelt majdnem két évtized óta a technológia nagymértékben fejlődött, habár az okmányok elolvasásának a fent ismertetetett alapvető lépései megmaradtak, csupán az azok végrehajtására használt megoldások változtak. Erre jó példa az optikai karakterfelismerés mély neurális hálókkal történő elvégzése, amely már az egészen kicsi, néhány rétegből álló hálók esetén is felülmúlja a hagyományos módszerek teljesítményét (Zhu–Ma–Feng–Dai 2018). A neurális hálók ugyanis meglehetősen hatékonyak bizonyos zavarok kezelésében (becsillanás, az okmányon lévő kosz és egyéb hatások okozta zajok). Érdemes megjegyezni, hogy az okmányok elolvasását és hitelesítését egybe lehet kötni a korábbi alfejezetben ismertetett arcfelismeréssel, ahogyan azt a repülőterekről ismert automatikus kiléptető gépekben is láthatjuk.

Érdemes megjegyezni, hogy a mély konvolúciós neurális hálók nemcsak nyomtatott, hanem írott karakterek elolvasásában is jeleskednek, tekintve, hogy ez volt az egyik első alkalmazásuk. Az írott karakterek felismerésében egy meglehetősen ígéretes új eredmény az úgynevezett kapszulaalapú háló (Mukhometzianov–Carrillo 2018) (Capsule Network), amely a konvolúciós architektúra kiváltását célozza. Ezek a hálók tanítása a konvolúciós hálóknál lényegesen költségesebb, és habár az írott karakterek felismerésében jobb eredményt értek el, bonyolultabb képosztályozás esetében az eredmények egyelőre nem kielégítők.

Egy utolsó fontos alkalmazás az aláírások verifikációja és a hamisítások detektálása, amelyre számos különböző módszer létezik. Az egyik módszer az úgynevezett fuzzy logikára alapszik, amely a hagyományos logikával szemben képes a részleges igazságokat is kezelni, így valós problémák esetén lényegesen jobban használható. A másik, elterjedtebb módszer nem meglepő módon a neurális hálózatok alkalmazása. Az aláírások hitelesítésére kifejlesztett módszerek közül az egyik leghatékonyabb Shah et. al (Shah–Sanghavi–Shah 2013) módszere, amely az aláírás görbéire illeszthető polinomok vizsgálatának alapján működik. Ezen polinomok együtthatóit egy neurális háló bemeneteként használták, és az így betanított hálózat a hamis aláírások 2%-át fogadta el helyesként, míg az eredeti aláírások 5%-át utasította el.

4.3. Virtuális és kiterjesztett valóság

A virtuális és kiterjesztett valóság rendszereknek számos alkalmazása van közigazgatási területeken. Ezek közé tartozik a különböző szimulációs tréningek megvalósítása, valamint a távoli kooperatív munkavégzés lehetőségének megteremtése. Tekintve, hogy ezek ismertetése a jelen kismonográfiának nem tárgya, ezért ezeket mindössze röviden említjük, míg a diszkusszió fókuszában az ilyen rendszerekben felhasznált számítógépeslátás-technológiákat tartjuk.

Hagyományos virtuális valóságrendszerek esetén a legtöbb esetben szükség van a bementi eszközként használt tárgy pozíciójának és orientációjának minél pontosabb követése, amelyet gyakran a követés és a háromdimenziós rekonstrukció módszereinek együttesének segítségével végeznek el. Hasonlóan esszenciális a felhasználók fejének minél pontosabb követése, ugyanis amennyiben a felhasználó által látott vizuális tartalom és a felhasználó mozgásérzete nincs tökéletes egyetértésben, annak könnyedén rosszullét lehet az eredménye.

Gyakori megoldás kiterjesztettvalóság-felületek esetében, hogy a felhasználó valamilyen valós tárgy segítségével képes a virtuális tartalommal interakcióba lépni, ez esetben

tapintható felhasználói felületek egy fajtájáról beszélhetünk. Ilyen felületek esetében a felhasználó egy, a tervező által előre elhelyezett valós tárgyat manipulál, a rendszer pedig a virtuális környezetet ezen a tárgy tulajdonságainak (pozíció, orientáció stb.) megváltozása alapján vezérli. Ezen felületek nagy előnye, hogy használatuk rendkívül természetes, így a felület könnyen megérthető és tanulható.

E megoldásnak azonban alapvető problémája, hogy a működésükhöz szükséges, hogy a munkaterület, illetve a felhasznált valós tárgyak a fejlesztők számára előre ismertek legyenek. Saját kutatómunkánk során egy olyan algoritmust alkottam meg, amely képes egy előre ismeretlen munkakörnyezet feltérképezésére, majd az adott virtuális objektumok és a környezetben található valós objektumok intelligens összepárosítására, vagyis a kiterjesztettvalóság-környezet „berendezésére".

Az algoritmus első lépéseként egy egykamerás, háromdimenziós rekonstrukciót végzünk, amelynek segítségével az előre ismeretlen munkaterület struktúrájáról információt nyerünk. Ezt követően a meglévő háromdimenziós jelenetben a RANSAC-algoritmus egy speciális variációjának segítségével különböző primitív formákat (sík, gömb, henger, kúp és tórusz) keresünk, majd ezekből egy gráfot konstruálunk. A konstruált gráf csomópontjai maguk a primitívek, az élek pedig az egyes formák közti geometriai kapcsolatot írják le. Ebben a reprezentációban az alapprobléma megfogalmazható úgy, mint az egyes virtuális objektumokhoz legjobban illő részgráfok kiválasztása a teljes munkaterület geometriáját leíró gráfból (Szemenyei–Vajda 2017).

Minden gráf csomóponthoz tartozik egy leíró vektor, amely az adott geometriai forma alapvető tulajdonságait írja le, így ezek segítségével könnyedén végezhetünk tanuló osztályozást, amelynek segítségével a gráf csomópontjait az egyes virtuális objektumoknak megfelelő kategóriákba oszthatjuk. Ennek a megközelítésnek viszont alapvető hátránya, hogy nem veszi figyelembe az egyes csomópontok közeli szomszédjait, vagyis azok lokális kontextusát. Ez azonban rendkívül fontos lenne, ugyanis az egyes primitív formák önmagukban ritkán alkotnak egy objektumot, ezek rend szerint több közeli primitívből állnak.

Ennek a problémának a megoldásához két külön módszert alkottunk: az első képes a gráfok csomópontjaihoz olyan sokdimenziós leíró vektorokat alkotni, melyek a csomópont tulajdonságai mellett a közeli csomópontokat is magukba foglalják, a központi csomóponttól való távolság mértékétől függő súllyal. A másik módszer az SVM-osztályozó esetén használatos véletlen séta kernel függvényt módosítja úgy, hogy ne teljes gráfok, hanem gráfok csomópontjainak hasonlósági függvényeként lehessen értelmezni (Szemenyei–Vajda 2018).

Az első módszer egyik hátránya, hogy a kapott leíró dimenziószáma meglehetősen nagy, amely az algoritmus számításigényét nagymértékben megnöveli. Éppen ezért célszerű egy olyan dimenzióredukciós technika alkalmazása, amely során az egyes osztályok elkülönítésére használható információ a redukció után is megmarad (például a korábban tárgyalt LDA-módszer). A jelen probléma során azonban egyetlen objektum egy több csomópontból álló gráf, ahol minden csomóponthoz tartozik egy külön leíró vektor. Az egyes csomópontok megkülönböztetése hasznos lehet pozíció és orientáció becslése esetén. Ezen megfontolás alapján egy saját módszert, az úgynevezett strukturált kompozit diszkriminancia analízist használjuk, amely ezt a kettős kritériumot igyekszik minél hatékonyabban teljesíteni (Szemenyei–Vajda 2017).

A csomópontok egyesével történő osztályozásának legnagyobb problémája, hogy a döntésnél a szomszédos csomópontok tulajdonságait ugyan figyelembe vesszük, a kapott címkéiket viszont nem. Ennek eredményeként globális értelemben szuboptimális végső elrendezések alakulhatnak ki, amelyek elkerülésére egy globális optimalizáló eljá-

rást érdemes alkalmazni. Munkánkban egy genetikus algoritmus segítségével kerestük meg azt az elrendezést, amely a lehető legjobb mértékben igazodik a csomópontok egyedi osztályozáshoz, azonban ezenfelül egyéb követelményeket is kielégít, mind például azt, hogy a közeli (nagy valószínűséggel ugyanahhoz a valós objektumhoz tartozó) csomópontok címkéje legyen azonos. Ezenfelül a genetikus algoritmus konvergenciáját saját fejlesztésű, feladatspecifikus operátorok alkalmazásával segítettük (Szemenyei –Vajda 2018).

4.4. Egyéb rendszerek

Fontos még említést tenni a számítógépes látórendszerek olyan közigazgatás szempontjából releváns alkalmazásairól, amelyek a korábban tárgyalt három kategóriába nem sorolhatók be. Ezek közül az egyik rendkívül lényeges az automatikus környezetértékelés területe. A környezetvédelem és a fenntartható fejlődés ugyanis korunk egyik legnagyobb kihívása, amelynek kezelésénél érdemes a modern technológiák adta lehetőségeket minél jobban kihasználni. A széles körben elérhető műholdas és légi felvételek segítségével számos környezeti jelenség megfigyelhető, a felvételek óriási mennyisége miatt azonban ezeknek emberi kiértékelése nem kivitelezhető.

Itt azonban nem csupán hagyományos, látható fénytartományában készített képek állnak rendelkezésre, hanem a legtöbb esetben több különböző tartomány is felhasználható. Ezek – a teljesség igénye nélkül – radar, LIDAR és infravörös (hő)képek lehetnek (Vorovencii 2011). Ezek együttesen felhasználhatók komplex környezetértékelési vagy környezeti hatásvizsgálati feladatok elvégzésére, melynek eredményei aztán különböző döntések alapjául szolgálhatnak. Ezek során a nyers felvételeken először a kamera földfelszínhez képesti pozíciójának ismeretében geometriai korrekciót kell végezni. Ehhez voltaképpen a kamera korábban tárgyalt külső paramétereire van szükség. Ezt követően különböző osztályozási módszerek segítségével meghatározhatjuk a képeken a különböző, döntés szempontjából releváns területfajtákat (erdő, lakott terület, megművelt terület, álló- és folyóvizek stb.).

Érdemes megjegyezni, hogy ilyen felvételek használhatók más célokra is, amelyek között kiemelkedő szerepet tölt be a mezőgazdaság, valamint a városfejlesztés. Torres et. al. (Torres, Pena-Barrag, Lopez-Granados, Jurado-Exposito, & Fernandez-Escobar, 2008) távoli felvételek segítségével olajfaültetvények állapotának automatikus kiértékelésére fejlesztett ki eljárást, míg Hormese és Saravanan (Hormese–Saravanan 2015) műholdképeket használ fel közutak automatikus detektálására, amelynek segítségével a térképek készítése automatizálható. Érdemes megjegyezni, hogy légi felvételekből származó képsorozatok segítségével háromdimenziós rekonstrukció is végezhető, amelynek segítségével domborzati és térbeli térképek is készíthetők.

5. ÖSSZEFOGLALÁS

A számítógépes látás algoritmusai a számítástudomány egyik rohamosan fejlődő és számtalan módon és helyen alkalmazható területe. *A számítógépes látórendszerek az e-közigazgatásban* címre hallgató kismonográfia két kötetében kísérletet tettem arra, hogy ennek a tématerületnek a legfontosabb eredményeit, megoldásait és közigazgatási alkalmazásait összefoglaljam, és ezzel egy, az olvasó számára hasznos ismereteket tartalmazó írást készítsek. A kismonográfia első kötetében a hagyományos számítógépes látás alapvető feladatait és az azokra megoldást kínáló algoritmusokat foglaltam össze röviden, kitérve azok alkalmazásaira is.

A jelenlegi, második kötet célja, hogy a számítógépes látás összetettebb, jelenleg is intenzív kutatás alatt álló részterületeibe adjon betekintést, valamint hogy ezek alkalmazásának legfontosabb praktikáit az olvasó számára átadja. Ezenfelül a kötet végén ismertettem a számítógépes látás segítségével megoldható közigazgatási jellegű feladatokat, és a jelenleg alkalmazott erre szakosodott látórendszerekbe is betekintést adtam. Fontos azonban megjegyezni, hogy a mesterséges intelligencia módszerein alapuló látórendszerek alkalmazása jelenleg gyerekcipőben jár, így az ezen módszerek által nyújtott lehetőségek tárháza a jövőben várhatóan bővülni fog.

HIVATKOZÁSOK

Bashbaghi, S. – Granger, E. – Sabourin, R. – Parchami, M. (2018): Deep Learning Architectures for Face Recognition in Video Surveillance. *arXiv.*

Bentley, J. L. (1975): Multidimensional Binary Search Trees Used for Associative Searching. *Coomunications of the ACM, 18*(9). 509–518.

Chan, E. – Panchanathan, S. (1993): Review of block matching based motion estimation algorithms for video compression. *Proceedings of Canadian Conference on Electrical and Computer Engineering.* Vancouver.

Cortes, C. – Vapnik, V. N. (1995): Support-vector network. *Machine Learning, 20*(3). 273–297.

Cui, L. – Li, K. – Chen, J. – Li, Z. (2011): Abnormal Event Detection in Traffic Video Surveillance Based on Local Features. *4th International Congress on Image and Signal Processing.*

Fischler, M. A. – Bolles, R. C. (1981): Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM, 24*(6). 381–395.

Girshick, R. (2015): Fast R-CNN. *arXiv.*

Girshick, R. – Donahue, J. – Darrell, T. – Malik, J. (2014): Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. *IEEE Conference on Computer Vision and Pattern Recognition.*

Goodfellow, I. J. – Pouget-Abadie, J. – Mirza, M. – Xu, B. – Warde-Farley, D. – Ozair, S. – Bengio, Y. (2014): Generative Adversarial Networks. *arXiv.*

Goodfellow, I. – Bengio, Y. – Courville, A. (2016): *Deep Learning.* MIT Press.

Grace, I. – Reshmi, K. (2015): Face Recognition in Surveillance System. *IEEE Sponsored 2nd International Conference on Innovations in Information, Embedded and Communication systems.*

Harris, C. – Stephens, M. (1988): A combined corner and edge detector. *Proceedings of the 4th Alvey Vision Conference.* 147–151.

He, K. – Gkioxari, G. – Dollár, P. – Girshick, R. (2017): Mask R-CNN. *arXiv.*

He, K. – Zhang, X. – Ren, S. – Sun, J. (2015): Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *IEEE International Conference on Computer Vision.*

He, K. – Zhang, X. – Ren, S. – Sun, J. (2016): Deep Residual Learning for Image Recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* Las Vegas, NV, USA.

Hirschmüller, H. (2008): Stereo Processing by Semi-Global Matching and Mutual Information. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 30*. 328–341.

Hochreiter, S. – Schmidhuber, J. (1997): Long short-term memory. *Neural Computation, 9*(8). 1735–1780.

Hormese, J. – Saravananb, C. (2015): Automated Road Extraction From High Resolution Satellite Images. *International Conference on Emerging Trends in Engineering, Science and Technology.*

Hossen, M. K. – Tuli, S. H. (2016): A Surveillance System Based on Motion Detection and Motion Estimation using Optical Flow. *5th International Conference on Informatics, Electronics and Vision (ICIEV).*

Huang, Q. – Zhou, K. – You, S. – Neumann, U. (2018): Learning to Prune Filters in Convolutional Neural Networks. *arXiv.*

Ibrahim, N. K. – Kasmuri, E. – Jalil, N. A. – Norasikin, M. A. – Salam, S. – Nawawi, M. R. (2013): License Plate Recognition (LPR): A Review with Experiments for Malaysia Case Study. *The International Journal of Soft Computing and Software Engineering, 3*(3). 83–93.

Ioffe, S. – Szegedy, C. (2015): Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv.*

James, G. – Witten, D. – Hastie, T. – Tibshirani, R. (2009): *An Introduction to Statistical Learning.* Springer.

Kató, Z. – Czúni, L. (2011): *Számítógépes látás.* Typotex Kiadó.

Kingma, D. P. – Welling, M. (2013): Auto-Encoding Variational Bayes. *arXiv.*

Le Cun, Y. (1988): A Theoretical Framework for Back-Propagation.

Lima, J. P., & Teichrieb, V. (2016). An Efficient Global Point Cloud Descriptor for Object Recognition and Pose Estimation. *29th Conference on Graphics, Patterns and Images (SIBGRAPI).* Sao Paulo, Brazil.

Lladbs, J. – Lumbreras, F. – Chapaprieta, V. – Queralt, J. (2001): ICAR: Identity Card Automatic Reader. *Sixth International Conference on Document Analysis and Recognition.*

Loshchilov, I. – Hutter, F. (2017): SGDR: Stochastic Gradient Descent with Warm Restarts. *International Conference on Learning Representations.*

Mukhometzianov, R. – Carrillo, J. (2018): CapsNet comparative performance evaluation for image classification. *arXiv.*

Redmon, J. – Divvala, S. – Girshick, R. – Farhadi, A. (2016): You Only Look Once: Unified, Real-Time Object Detection. *IEEE Conference on Computer Vision and Pattern Recognition.*

Ren, S. – He, K. – Girshick, R. – Sun, J. (2017): Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 39*(6). 1137–1149.

Ruder, S. (2016): An overview of gradient descent optimization algorithms. *arXiv.*

Rusu, R. B. – Blodow, N. – Beetz, M. (2009): Fast Point Feature Histograms (FPFH) for 3D registration. *IEEE International Conference on Robotics and Automation.* Kobe, Japan.

Schnabel, R. – Wahl, R. – Klein, R. (2007): Efficient RANSAC for Point□Cloud Shape Detection. *Computer Graphics Forum.*

Shah, V., Sanghavi, U., & Shah, U. (2013). Off-line Signature Verification Using Curve Fitting Algorithm with Neural Networks. *International Conference on Advances in Technology and Engineering (ICATE).*

Shahad, R. A. – Bein, L. G. – Saad, M. H. – Hussain, A. (2016): Complex Event Detection in an Intelligent Surveillance System using CAISER Platform. *International Conference on Advances in Electrical, Electronic and System Engineering.* Putrajaya, Malaysia.

Shelhamer, E. – Long, J. – Darrell, T. (2017): Fully Convolutional Networks for Semantic Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 39*(4). 640–651.

Srivastava, N. – Hinton, G. – Krizhevsky, A. – Sutskever, I. – Salakhutdinov, R. (2014): Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research, 15*. 1929–1958.

Sultani, W. – Chen, C. – Shah, M. (2018): Real-world Anomaly Detection in Surveillance Videos. *arXiv.*

Sun, J. – Zheng, N.-N. – Shum, H.-Y. (2003): Stereo matching using belief propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 25*(7). 787–800.

Szegedy, C. – Liu, W. – Jia, Y. – Sermanet, P. – Reed, S. – Anguelov, D. – Rabinovich, A. (2015): Going Deeper with Convolutions. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* Boston, MA, USA.

Szeliski, R. (2010): *Computer Vision: Algorithms and Applications.* Springer Science and Business Media.

Szemenyei, M. – Vajda, F. (2017): Dimension reduction for objects composed of vector sets. *International Journal of Applied Mathematics and Computer Science, 27*(1).

Szemenyei, M. – Vajda, F. (2018): 3D Object Detection and Scene Optimization for Tangible Augmented Reality. *Periodica Polytechnica Electrical Engineering and Computer Science, 62*(8). 25–37.

Taketomi, T. – Uchiyama, H. – Ikeda, S. (2017): Visual SLAM algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications, 9*(16). 1–11.

Torres, L. G. – Pena-Barrag, J. – Lopez-Granados, F. – Jurado-Exposito, M. – Fernandez-Escobar, R. (2008): Automatic assessment of agro-environmental indicators from remotely sensed images of tree orchards and its evaluation using olive plantations. *Computers and electronics in agriculture, 61*. 179–191.

Upasana, A. – Manisha, B. – Mohini, G. – Pradnya, K. (2015): Real Time Security System using Human Motion Detection. *International Journal of Computer Science and Mobile Computing, 4* (11). 245–250.

Vorovencii, I. (2011): Satellite Remote Sensing in Environmental Impact Assessment: an Overview. *Bulletin of the Transilvania University of Braşov, 4* (53). 73–80.

Wang, P. – Chen, P. – Yuan, Y. – Liu, D. – Huang, Z. – Hou, X. – Cottrell, G. (2017): Understanding Convolution for Semantic Segmentation. *arXiv.*

Wu, C. (2013): Towards Linear-time Incremental Structure From Motion. *3DV.*

Wu, C. – Agarwal, S. – Curless, B. – Seitz, S. M. (2011): Multicore Bundle Adjustment. *CVPR.*

Zhu, J. – Ma, H. – Feng, J. – Dai, L. (2018): ID card number detection algorithm based on convolutional neural network. *AIP Conference Proceedings.*

A Nemzeti Közszolgálati Egyetem kiadványa

Kiadó:

Nemzeti Közszolgálati Egyetem
Közigazgatási Továbbképzési Intézet
www.uni-nke.hu

Felelős kiadó:
Prof. Dr. Kis Norbert rektorhelyettes
Címe: 1083 Budapest, Üllői út 82.

Olvasószerkesztő:
Dorogi Katalin

Tördelőszerkesztő:
Friebert Máté

ISBN 978-963-498-116-9 (elektronikus)