Eötvös Loránd Tudományegyetem
Informatikai Kar
Információs Rendszerek Tanszék

Szemdetektor

Kiszlinger Melinda

Témavezető:

dr. habil Lőrincz András

Budapest, 2005. június 17.

Tartalomjegyzék

1. fejezet

Bevezetés

Dolgozatomban a Viola-Jones objektum detektálással, a detektorok tanuló algoritmusával készített szemdetektorokkal és az ELTE NIPG csoportjában fejlesztés alatt álló szemkövető programmal foglalkozom.

Paul Viola és Michael J. Jones robosztus valós-idejű objektum detektáló rendszerének nagysága a gyors képfeldolgozási képességében és magas detektálási arányában rejlik. Három fontos tulajdonsága van. Első a képreprezentáció, melyet *integrális képnek* nevezünk. Ez a reprezentáció biztosítja a detektor által használt *jellemzők* (features) nagyon gyors kiszámítását. Második fontos tulajdonsága az *AdaBoost-on alapuló tanuló algoritmus.* Ez az algoritmus kiválasztja a kritikus jellemzők kis részhalmazát, melyekkel hatékonyan osztályozhatunk. Harmadik fontos tulajdonsága *az osztályozók kaszkádba szervezése, sorba rendezése.* Ezek a kaszkádok gyorsan megtalálják a kép háttér területeit, melyen keresett objektum nem fordul elő, és a sokat ígérő részekkel töltenek több számítási időt.

A Viola-Jones detektort arcdetektálásra használták elsősorban készítői. A tanuló algoritmus felhasználásával szemdetektorokat fejlesztettem, szemkörnyék, balszem és jobbszem detektorokat.

A detektorokat az ELTE NIPG csoportjában fejlesztett szemkövető program pontosságának növelésére használtam fel. A szemkövető program kézi- illetve webkamera képen követi a felhasználó szemgolyó köreit. A Viola-Jones detektorok segítségével szűkítem a keresési területet, így gyorsítva a program futást. A programba beépített detektor hierarchia arc, szemkörnyék, balszem és jobbszem detektorokat használ. Egy 640x480 képpont felbontású webkamera képén a teljes detektor hierarchiát futtatva minden beérkező képen 10 kép/másodperc sebességet értem el egy 1.6 GHz-es Intel Pentium M processzorral rendelkező notebook-on. Ha minden ötödik képen végzünk csak detektálást, ami a program futásához tökéletesen elég, akkor 20 kép/másodperc detektálási sebességet érhetünk el a teljes detektor hierarchiát futtatva. A szemkövető program C++ programnyelven Microsoft Windows környezetben készül.

A szemkövető program távlati célja egy tekintet követő rendszer fejlesztése. A tekintet követéshez jelenleg speciális eszközöket használnak, melyek pontosak, ugyanakkor meglehetősen drágák. Ilyen speciális eszköz például az infra kamera, ami az infra megvilágítás során a szemgolyón létrejövő csillanásokból határozza meg a tekintet irányát.

Egy 640x480 képpont felbontású webkamera képen a felhasználó szemgolyóinak sugara átlagosan 10 képpont, az így nyert információ tehát igen kevés. A pontosság növelésére a mesterséges intelligencia eszközeire lehet szükség. A szemkövető program jelenlegi verziójában mesterséges neuronhálókat használ fel a szemgolyó öt irányultságának, a balra, középre, jobbra, felfele és lefele néző szempozíciók meghatározására. A neuronhálók a Viola-Jones által is használt jellemzők értékeit dolgozzák fel. A szempozíciók meghatározása még távol van a tekintet követéstől, de már felhasználhatjuk irány utasításokhoz, azaz nyíl-billentyű események generálásához, vagy definiálhatunk hozzájuk szemgesztusokat, például gyors jobb-bal-jobb szemmozgást.

A tekintet követés az ember-számítógép interakció egyik fontos eszköze. Felhasználható például mozgáskorlátozott emberek kommunikációjában, a beviteli hatékonyság fokozásában, katonai alkalmazásokban, játékokban, mobil eszközökben.

1.1. Tézisek

1. A Viola-Jones objektum detektorok robosztus valós-idejű objektum detektálásra képesek. Érzéketlenek, azaz robosztusok a beérkező kameraképre, gyors feldolgozásra képesek magas detektálási arány mellett. Egy 30 kép/másodperc teljesítményű 640x480 képpont felbontású webkamera képén, a Viola-Jones arcdetektor 30 kép/másodperc teljesítményt ér el egy 1.6 GHz-es Intel Pentium M processzorral rendelkező notebook-on, tehát képes az összes beérkező kép feldolgozására.

2. Detektorok tanítása során a következő paraméterek befolyásolják a detektálási és a hamis találati arányt: a szimmetria beállítása, a pozitív példák mérete, száma, és a negatív példák száma. A detektálási arány és a hamis találati arány csökken a szimmetria hamisra állításával, a pozitív példák méretének csökkentésével, számuk növelésével, és a negatív példák számának növelésével.

3. Az ELTE NIPG csoportjában fejlesztés alatt álló szemkövető programban hatékonyan csökkenthetjük a szemgolyó körök keresési területét a Viola-Jones arc, szemkörnyék, balszem és jobbszem detektorok segítségével.

4. A Hough körkereséssel jó eredményeket kapunk, amennyiben a felhasználó teljes szemgolyója látszik. Ha ez nem teljesül, például mert a szemgolyókból sok embernél csak egy „csík” látszik, vagy mert mosolyog/nevet a felhasználó, akkor a körkeresés helyett körrészlet keresésre van szükség, vagy egy másik lehetséges megoldás az összefüggő komponensek keresése.

5. Mesterséges neuronhálók segítségével irány utasításokat, azaz nyíl-billentyű eseményeket generálhatunk a szemkövető programmal. Meghatározható, hogy a felhasználó a monitor melyik felébe néz, balra, középre, jobbra, felfele vagy lefele.

1.2. A dolgozat felépítése

A 2. fejezet a Viola-Jones objektum detektálás elméleti hátterét mutatja be. A 3. fejezet a szemdetektorok tanításának optimalizálását ismerteti az Intel Open CV könyvtárban implementált tanuló algoritmus paramétereinek függvényében. A 4. fejezet az ELTE NIPG csoportjában fejlesztés alatt álló szemkövető programról szól.

A Függelék 7.1. fejezete foglalkozik az Viola-Jones arcdetektor eredményeivel. A 7.2. fejezet az általam tanított szemdetektorok táblázatos teszteredményeit tartalmazza. A 7.3. fejezetben a Hough transzformációról olvashatunk, melyet a szemkövető program a szemgolyó körök keresésre használ. A 7.4. fejezet mesterséges neuronhálókról szól, melyeket a szemkövető program irány utasításaihoz használok fel.

2. fejezet

Viola-Jones objektum detektálás

Paul Viola és Michael J. Jones robosztus valós-idejű objektum detektort fejlesztettek. A rendszer robosztussága abban rejlik, hogy érzéketlen a feldolgozandó kép minőségére. Kialakítottak egy frontális arcdetektor rendszert, mely eléri az előtte publikált legjobb eredmények [2, 3, 4, 5, 6] *találati és hamis pozitív arányát* (2.0.4. és 2.0.5. definíció). Az arcdetektor mintájára szemdetektorokat fejlesztettem az Intel Open CV[1] szabadon felhasználható képfeldolgozó-könyvtár segítségével, melyben Viola és Jones közzétették arcdetektorukat és az implementált tanítási algoritmust.

Arcdetektáló rendszerük tisztán kiemelkedik az eddigi megközelítések közül gyorsaságában. Valós időben 30 kép/másodperc teljesítményt érhetünk el, 640x480 pixel felbontású kameraképen egy 1.6 GHz-es Intel Pentium M processzorral rendelkező notebook-on. Más arcdetektáló rendszerekben, kisegítő információkat használnak, hogy magas képfeldolgozási arányt érjenek el. Ilyen például egymás utáni képek különbsége videó filmeken vagy pixel-szín színes képeken. A Viola-Jones detektor szürkeskálás képekből kapott információk alapján képes magas képfeldolgozási arányt elérni. Az említett egyéb információ források is integrálhatók a Viola-Jones rendszerbe, így még magasabb detektálási arányt érhetünk el. A következő definíciók a további részek érthetősége miatt kerültek ide.

2.0.1. Definíció. ***Hamis találat vagy hamis pozitív****: Ha a detektor a kép egy részletén igaz eredménnyel tér vissza, miközben a képrészleten nem szerepel a keresett objektum, akkor azt hamis találatnak vagy hamis pozitívnak nevezzük.*

2.0.2. Definíció. ***Hamis negatív****: A nem detektált objektumot hamis negatívnak nevezzük.*

2.0.3. Definíció. ***Találat****: Ha a detektor a kép egy részletre igaz eredményt ad, és a képrészlet a keresett objektumot valóban ábrázolja, akkor ezt az eseményt találatnak nevezzük.*

2.0.4. Definíció. ***Hamis pozitív arány vagy hamis találati arány****: Detektor tesztelése során a hamis találatok számát osztva a vizsgált képrészletek számával kapjuk a hamis pozitív arányt vagy hamis találati arányt. Ez egy valós szám a* [0..1] *intervallumból.*

[1] `http://sourceforge.net/projects/opencvlibrary/`

2.0.5. Definíció. ***Találati vagy detektálási arány****: Detektor tesztelése során a találatok számát osztva a keresett objektumok számával kapjuk a találati vagy detektálási arányt. Ez egy valós szám a* [0..1] *intervallumból.*

Három fontos tulajdonsága van a Viola-Jones objektum detektáló rendszernek. Most röviden bevezetem, majd a későbbi szakaszokban részletesebben kifejtem ezeket.

Az első fontos tulajdonság az új képreprezentáció, melyet *integrális képnek* nevezünk. Az integrális kép segítségével nagyon gyors *jellemző* (feature) kiértékelésre vagyunk képesek. Részben Papageorgiou és társai munkája [7] által motiváltan a Viola-Jones rendszer nem közvetlenül a képintenzitásokkal dolgozik, hanem képrégiók jellemzőivel. Hasonlóan Papageorgiou-hoz, jellemzők egy halmazát használja, melyek emlékeztetnek a Haar-féle bázisfüggvényekre. Az integrális kép képpontonként néhány elemi művelettel számítható ki a képből. Az integrális kép ismeretében pedig a Haar jellemzők konstans időben számolhatók minden skálán és pozícióban.

A második fontos tulajdonság egy osztályozó létrehozásának folyamata, a fontos jellemzők egy kis halmazának kiválasztásával, az AdaBoost-on alapuló tanuló algoritmust használva [8]. Egy kép részablakán belül az összes Haar jellemzők száma nagyon nagy, sokkal nagyobb, mint a pixelek száma. Azért, hogy a gyors osztályozást biztosítsuk, a tanuló folyamatnak ki kell zárnia az elérhető jellemzők nagy többségét, és a kritikus jellemzők kis halmazára kell koncentrálnia. Tieu és Viola munkája által motiváltan, a jellemző kiválasztás az AdaBoost egy egyszerű módosítása lett: minden gyenge osztályozót kényszerítenek, hogy annak eredménye csak egy jellemzőtől függjön [9]. Ennek eredményeként a Boost folyamat minden köre, amely új gyenge osztályozót választ, megfelel egy jellemző kiválasztó folyamatnak [10, 11, 7].

A harmadik fontos tulajdonság több bonyolultabb osztályozó egymás után kombinálása, *kaszkád struktúrába szervezése, sorba rendezése*, mely lényegesen növeli a detektor sebességét, segítségével a kép sokat ígérő területeire tudunk fókuszálni. Egy képen gyakran gyorsan meghatározható, hol fordulhat elő egy objektum [12, 13, 6]. Még összetettebb folyamatokat a sokat ígérő részek számára tartunk fenn. Legfőbb mércéje egy ilyen megközelítésnek a vizsgálati folyamat hamis negatív aránya, azaz a nem detektált objektumok aránya. Ezért a figyelmi szűrőnek mindegyik vagy majdnem mindegyik objektumot ki kell választania.

Láthatjuk majd egy nagyon egyszerű és hatékony osztályozó tanulási folyamatát, amely "felügyelt" figyelmi szűrőként használható. A felügyelt kifejezés arra a tényre utal, hogy a figyelmi operátort úgy tanítjuk, hogy sajátos osztályok példányait detektálja. Az arcdetektálás terén lehetséges, hogy kevesebb, mint 1% hamis negatív, észrevétlen objektum arányt, és 40% hamis pozitív, ál-objektum arányt érjünk el egyetlen osztályozót, figyelmi szűrőt használva, mely 20 egyszerű művelettel kiértékelhető (ez megközelítőleg 60 mikroprocesszor utasítás). Ennek a szűrőnek a hatása, hogy kevesebb, mint felére csökkenti azokat a területeket, ahol a végső detektort ki kell értékeljük.

Azok a kép-alablakok, melyeket a kezdeti osztályozó, vagyis a figyelmi szűrő nem utasít el, feldolgozásra kerülnek az osztályozók szekvenciájában, melyek minden egyede valamivel komplexebb az előzőnél. Ha az alablakot bármely osztályozó

elutasítja, az kiesik a feldolgozási folyamatból. A kaszkád detektálási folyamat lényegében egy degenerált döntési fa, és mint ilyen, kapcsolatban áll Amit és Geman munkájával [6].

A megvalósított arcdetektor kaszkádja 32 osztályból áll, mely több mint 80,000 műveletet jelent. Mindamellett a kaszkád struktúra átlag detektálási ideje kevés, így a detektor igen gyors. Egy bonyolult adathalmazon, mely 507 arcot és 75 millió alablakot tartalmaz, alablakonként átlag 270 mikroprocesszor utasítással detektáltak arcot. Összehasonlításul, a Viola-Jones rendszer 15-ször gyorsabb a Rowley és társai által implementált detektornál [3].

Egy gyors arcdetektornak széles alkalmazási köre alakulhat ki. A sebesség növekedése valós idejű arcdetekálást tesz lehetővé rendszereken, melyeken ez korábban kivitelezhetetlen volt. Azokban a rendszerekben, melyekben a gyorsaság nem lényeges, a Viola-Jones rendszer jelentős utófeldolgozást és elemzést engedélyez. Implementálható alacsony teljesítményű eszközök széles skáláján, beleértve például a kézikamerákat és integrált processzorokat is. A detektor készítői például egy Compaq iPaq kézikamerán implementálták rendszerüket, és azon 2 kép/másodperc mellett detektáltak arcot. (Az eszköznek egy alacsony teljesítményű 200 mips-es *Strong Arm* processzora volt, mely lebegő pontos számításra képes hardvert nélkülözött.)

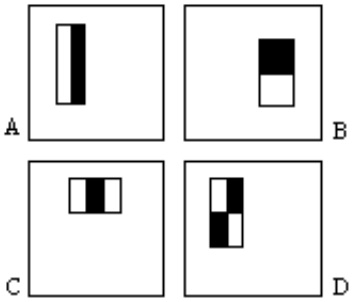
A továbbiakban a Viola-Jones detektor implementálásával, a kapcsolódó elméletekkel és kísérletekkel foglalkozom. A 2.1. fejezet a jellemzők alakjával foglalkozik, és azok gyors számításának kivitelezésével. A 2.2. fejezet a jellemzők osztályozók formájába szervezésével foglakozik. A tanulási folyamat, az AdaBoost egy variánsa, jellemző kiválasztó folyamatként működik. Míg az AdaBoost-tal konstruált osztályozók jó számítási és osztályozási teljesítménnyel rendelkeznek, addig valós idejű osztályozásra túlságosan lassúak, ezért volt szükség az AdaBoost tanuló algoritmus módosítására. A 2.3. fejezet foglalkozik olyan osztályozó kaszkád konstruálásával, azaz osztályozók sorba rendezésével, mely megbízható és hatékony objektum detektort eredményez. Az 7.1. fejezet a kísérletek számszerű leírásával foglalkozik, a szerzők kísérleti módszertanának részletezett leírását is beleértve.

2.1. Jellemzők

A Viola-Jones objektum detektáló rendszer egyszerű jellemzők értékei alapján osztályoz képeket. Sok érv szól a jellemzők használata mellett, a képpontok közvetlen használata helyett. A legáltalánosabb ok, hogy a jellemzők ad-hoc tudás területeket képesek kódolni, melyeket nehéz megtanulni véges mennyiségű tanító adathalmazt használva. A rendszer egy másik fontos motivációja a jellemzők használatára: a jellemző-alapú rendszer sokkal gyorsabban dolgozik, mint a képpont-alapú.

A használt egyszerű jellemzők emlékeztetnek a Haar-féle bázisfüggvényekre, melyeket Papageorgiou és társai használtak [7]. A Viola-Jones rendszer ezeknek a jellemzőknek három fajtáját használja. A *két-téglalap jellemző* értéke a pixelek összegének különbsége két téglalap terület között. A területeknek ugyanakkora a mérete és alakja, és függőlegesen vagy vízszintesen szomszédosak (2.1. ábra). A *három-téglalap jellemző* két szélső téglalap pixeleinek összegét vonja le a középső téglalap pixeleinek összegéből. Végül a *négy-téglalap jellemző* diagonális téglalap párok közötti különbséget számít.

Feltételezve, hogy a detektor alap felbontása 24x24-es, a teljes halmaza a téglalap jellemzőknek igen nagy, 45,396, a Haar-féle bázisfüggvényektől eltérően.



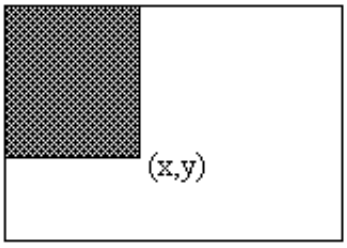
2.1. ábra. Téglalap jellemzők a detektor ablakon belül. A fehér téglalapon belül fekvő pixelek összegét vonjuk ki a fekete téglalapon belül fekvő pixelek összegéből. A és B mutatja a két-téglalap, C a három-téglalap, D pedig a négy-téglalap jellemzőt.

2.1.1. Integrális kép

A téglalap jellemzőket gyorsan ki lehet számolni a kép közbülső reprezentálását használva, melyet integrális képnek nevezünk. Az integrális kép az x, y pontban a tőle balra fent levő téglalapban elhelyezkedő pixelek intenzitásértékeinek összege:

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x',y') \tag{2.1}$$

ahol $ii(x,y)$ az integrális kép, és $i(x,y)$ az eredeti kép (2.2. ábra).



2.2. ábra. Az integrális kép értéke az (x,y) pontban a balra fent levő téglalapban elhelyezkedő pixelek intenzitásértékeinek összege.

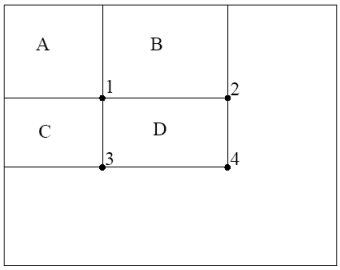
Használjuk a következő rekurzív párokat:

$$s(x,y) = s(x,y-1) + i(x,y) \tag{2.2}$$

$$ii(x,y) = ii(x-1,y) + s(x,y) \tag{2.3}$$

Ahol $s(x,y)$ az intenzitás értékek halmozódó sorösszege, $s(x,-1) = 0$, és $ii(-1,y) = 0$. Az integrális képet így egyszer kell csak számolni az eredeti képből, majd a későbbiekben a jellemző kiértékelésben használjuk.

Az integrális képet használva egy téglalap összeg négy tömbhivatkozással számítható (2.3. ábra). Két téglalap közötti különbség nyolc hivatkozással számítható. A két-téglalap jellemző mivel szomszédos téglalapok különbségét veszi, hat tömbhivatkozással számítható, a három-téglalap jellemző nyolc hivatkozással, a négy-téglalap jellemző kilenc hivatkozással.



2.3. ábra. A D-be eső képpontok összege az integrális kép négyszeri elérésével meghatározható. Az integrális kép értéke az 1-es pozícióban az A-ba eső képpontok intenzitásösszege. 2-ben az érték A + B, 3-ban A + C, 4-ben A + B + C + D. Így a D-be eső összeg a 4 + 1 - (2 + 3) összeüggés alapján számítható ki.

Egy alternatív motiváció az integrális képhez Simard munkája a „boxletekről" [14], melyben megállapítják, hogy lineáris művelet esetén (pl. $f \cdot g$), bármely invertálható művelet alkalmazható f-re vagy g-re, ha az inverzük alkalmazható az eredményre. Például konvolúció esetén, ha a derivált operátort alkalmazzuk a képre és a magfüggvényre, a konvolúció eredménye a deriváltak konvolúciójának második integráltja kell legyen:

$$f * g = \int\int (f' * g'). \tag{2.4}$$

A szerzők tovább mennek és megmutatják, hogy a konvolúció jelentősen gyorsítható, ha f és g deriváltja ritka, vagy azzá tehető. Egy hasonló meglátás, hogy egy invertálható lineáris művelet alkalmazható f-re, ha az inverze g-re alkalmazott:

$$(f'') * \left(\int\int g \right) = f * g. \tag{2.5}$$

A téglalap összeg számítás kifejezhető egy szorzásként, $i \cdot r$, ahol i a kép és r az elhatároló kép (amiben 1 az érték a figyelt téglalapon belül és 0 azon kívül). Ez a művelet a következőképpen írható:

$$i \cdot r = \left(\int\int i \right) \cdot r''. \tag{2.6}$$

Az integrális kép tulajdonképpen az eredeti kép második integrálja (először a sorok, majd az oszlopok mentén).

2.1.2. Jellemzők diszkussziója

A téglalap jellemzők valamivel egyszerűbbek összehasonlítva más alternatívákkal, mint például a steerable filter-ek [15, 16]. A steerable filter-ek és rokonaik kitűnően

alkalmazhatók a határok részletezett elemzésére, képtömörítésre, és textúra elemzésre. Ezzel szemben a téglalap jellemzők egészen durvák, habár érzékenyek az élek, vonalak és más egyszerű struktúrák jelenlétére. Eltérően a steerable filter-ektől a téglalap jellemzőkkel csak a horizontális és vertikális irányítások elérhetők. Mégis a téglalap jellemzők halmaza gazdag képreprezentációt nyújt, ami segíti a hatékony tanulást. A magas számítási hatékonyság bőségesen kompenzálja a korlátozott rugalmasságot.

Vizsgáljunk meg egy hagyományos szemléletet, amiben a kép piramisát számoljuk, azért, hogy egyszerűségük és látszólagos rugalmatlanságuk ellenére értékelni tudjuk a téglalap jellemzőket. Hasonlóan a legtöbb objektum detektáló rendszerhez a Viola-Jones detektor több skálán is végignézi a képet; az alap helyzetből indul, ahol az objektumokat 24x24 pixelben keresi, majd a képet 11 skálán tapogatja le, mindegyik skála növekedési faktora 1.25. A hagyományos megközelítés 11 piramisképet kiszámítása, ahol mindegyik kép 1.25-ször kisebb az előzőnél. Ezután egy fix méretű detektor tapogatja le a képeket. A piramis képek kiszámítása jelentős időt igényel. Hagyományos hardware használatával igen nehéz 15 kép/másodperc mellett kiszámolni a piramis képeket[2].

Ezzel szemben Viola és Jones definiálják a jellemzők egy jelentős halmazát, amelyek tulajdonában egy egyszerű jellemző bármely skálán és pozícióban pár művelettel kiszámítható. A 2.3. fejezetben látható, hogy hatékony arcdetektor létrehozható már néhány két-téglalap jellemző segítségével is. A jellemzők kiszámításának hatékonysága miatt az arcdetektor végigvihető egy teljes képen minden skálán és pozícióban 15 kép/másodperc mellett, kevesebb idő alatt, mint ami a 11 piramis kép kiszámításához kell. Ezért bármely detektor, amihez ki kell számolni a képek piramisát, lassabban fut a Viola-Jones arcdetektornál.

2.2. Osztályozó függvények tanítása

Ha adott a jellemzők, továbbá pozitív és negatív tanítópéldák egy-egy halmaza, a gépi tanulás bármely megközelítése használható egy osztályozó függvény tanulásához. Sung és Poggio a Gauss modell egy keverékét használják [2]. Rowley, Baluja és Kanada egyszerű jellemzők egy kis halmazát és neuronhálót használnak [3]. Osuna és mások SVM-et (support vector machine-t) használnak [11]. Roth és társai egy új és eddig nem használt képreprezentációt javasoltak és használtak a Winnow tanulási folyamattal együtt [5].

Már említettük, hogy egy kép-alablaknak 45,396 téglalap jellemzője van. Ez a szám sokkal nagyobb, mint a képpontok száma. Habár minden jellemző hatékonyan számítható, a jellemzők teljes halmazának számítása megengedhetetlen. Ezeknek a jellemzőknek már igen kicsi halmazával is létrehozható hatékony osztályozó. Az igazi kihívás ezeknek az osztályozók a megtalálása.

A Viola-Jones rendszerben az AdaBoost egy változatát használják mind a jellemző kiválasztásra, mind pedig az osztályozó tanítására. Eredeti formájában az AdaBoost gyenge osztályozó függvények kombinálásával képes egy erős osztályozót

[2]A képpontok száma 11 szinten körülbelül $55 * 384 * 288 = 6082560$. Feltéve, hogy minden pixel 10 műveletet igényel a kiszámításhoz, a piramis körülbelül 60,000,000 műveletbe kerül. Tehát körülbelül 900,000,000 művelet/másodperc kell a 15 kép/másodperces feldolgozáshoz.

előállítani, ezáltal tetszőleges tanuló algoritmus javítására használható. Az alapgondolat a következő: a gyenge osztályozót lefuttatjuk egy tanítóhalmazon. A teljes tanítóhalmaz osztályozása után az előző osztályozó által hibásan osztályozott példákat újra súlyozzuk, és újra elvégezzük a tanítást. Így a végső erős osztályozó egy perceptron lesz, a gyenge osztályozók lineáris kombinációja, kiegészítve egy küszöbértékkel.

Freund és Shapire bebizonyította, hogy az erős osztályozó tanulási hibája a körök számával párhuzamosan exponenciálisan tart nullához. Még fontosabbak az általános teljesítmény számeredményei, melyeket később bizonyították [10].

A hagyományos AdaBoost folyamatot könnyen értelmezhetjük egy mohó jellemző kiválasztó folyamatként. A fokozás általános problémáját figyelembe véve, amiben a jellemző osztályozó függvények egy nagy halmazát kombináljuk össze súlyozott többségi szavazás alapján, az igazi kihívás, hogy a jó osztályozókhoz nagy súlyt adjunk, míg a kevésbé jó függvényekhez kicsit. Az AdaBoost egy agresszív eljárás jó osztályozók kis halmazának kiválasztására, amik jelentősen eltérőek lehetnek. Egy analógiát rajzolva a gyenge osztályozók és a jellemzők közé, az AdaBoost egy hatékony eljárás jó jellemzők egy kis halmazának keresésére, melyek jelentősen eltérőek lehetnek.

Egy gyakorlati eljárás ennek az analógiának a kiegészítésére, hogy megszorítjuk a gyenge tanulót az osztályozó függvényeknek arra a halmazára, melyek csak egyszerű jellemzőktől függnek. Ennek eléréséhez a gyenge tanuló algoritmust úgy tervezték meg, hogy egy egyszerű téglalap jellemzőt válasszon, ami a legjobban szétválasztja a pozitív és negatív példákat (ez hasonló [9] megközelítéséhez a képvisszakeresés területén). Minden jellemzőhöz a gyenge tanuló meghatározza az optimális küszöb osztályozó függvényt, ami a példák minimális számú halmazát osztályozza rosszul. Így egy gyenge osztályozó ($h_j(x)$) egy jellemzőből (f_j), egy küszöbből (Θ_j) és egy paritásból (p_j) áll, ahol a paritás az egyenlőtlenség jel irányát jelöli:

$$h_j(x) = \begin{cases} 1 & \text{ha } p_j f_j(x) < p_j \Theta_j \\ 0 & \text{különben} \end{cases}$$

Itt x egy 24x24 pixel felbontású kép-alablak.

Gyakorlatban nem lehet egyszerű jellemzővel megvalósítani az osztályozást úgy, hogy alacsony hibát érjünk el. A folyamat korai szakaszában kiválasztott jellemzők 0.1 és 0.3 közötti hibaarányt adnak. A későbbi körökben választott jellemzők, melyek egyre bonyolultabbak, 0.4 és 0.5 közötti hibaarányt adnak.

A tanuló algoritmus leírása következik. Egy kérdés online megtanulásának menetét írja le a folyamat. Minden T hipotézis egy egyszerű jellemzőt használ. A végső hipotézis a T hipotézisek egy súlyozott lineáris kombinációja, ahol a súlyok fordítottan arányosak a tanulási hibával.

- Adottak n darab tanítóminta $(x_i, y_i), \ldots, (x_n, y_n)$, ahol $y_i = 0, 1$ a negatív illetve pozitív képeknél.
- A kezdeti súlyok $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ minden $y_i = 0, 1$ -nek megfelelően, ahol m és l a negatív illetve pozitív példák száma.
- Minden $t = 1, \ldots, T$-re (T darab gyenge osztályozót kombinálunk össze)

1. Normalizáljuk a súlyokat úgy, hogy w_t a valószínűségi eloszlás legyen,
$$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,i}}.$$
2. Minden j jellemzőre tanítsunk egy h_j osztályozót, ami csak egyetlen egyszerű jellemzőt használhat. A w_t-vel súlyozott hiba $\epsilon_j = \sum_i w_i |h_j(x_j) - y_i|$.
3. Válasszuk ki h_t osztályozót, melynek a legkisebb ϵ_t hibája van.
4. Súlyozzuk át a tanítóhalmazt a következőképpen:
$$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i},$$
ahol $e_i = 0$ ha az x_i példát helyesen osztályoztuk, $e_i = 1$ különben, és $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
5. A végső erős osztályozó az így kapott gyenge T darab h_t osztályozó kombinációja:
$$h(x) = \begin{cases} 1 & \sum_{t-1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t-1}^{T} \alpha_t \\ 0 & \text{különben} \end{cases}$$
ahol $\alpha_t = \log \frac{1}{\beta_t}$.

2.2.1. Tanítás diszkussziója

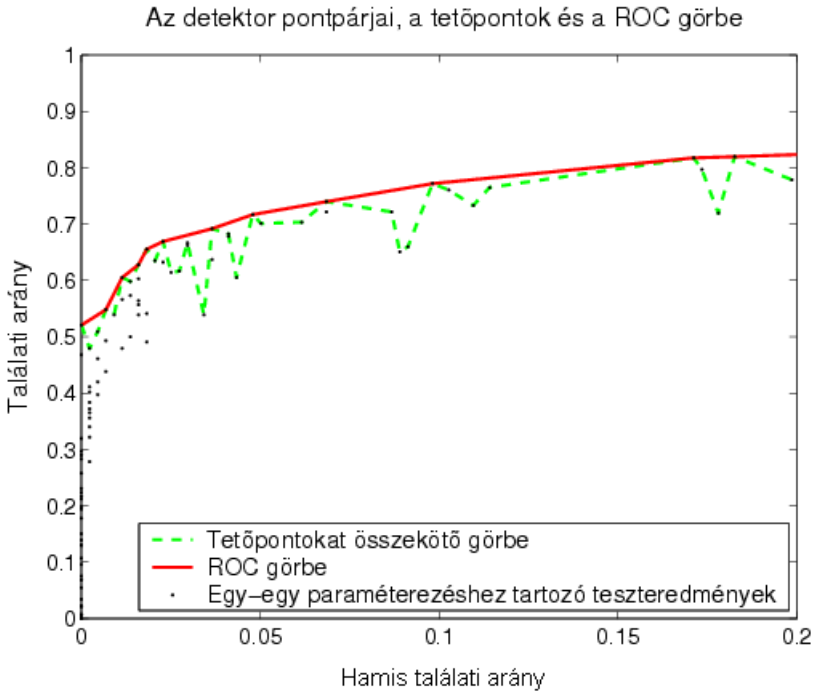
Több általános jellemző kiválasztó folyamatot is javasoltak már (lásd [17] 8. fejezetét). A Viola-Jones féle végső alkalmazás egy agresszív folyamat, ami a jellemzők túlnyomó többségét ki kell dobja. Egy hasonló felismerési problémához Papageorgiou és társai például egy a jellemzők eltérésein alapuló jellemző kiválasztó folyamatot javasoltak [7]. Ez a folyamat jó eredményt hozott, 37 jellemzőt választott ki 1734 közül. Míg ez egy jelentős csökkenés, addig a minden alablakra kiértékelt jellemzők száma még mindig meglehetősen nagy.

Roth és társai egy a Winnow exponenciális perceptron tanulási szabályokra épülő jellemző kiválasztó folyamatot javasol [5]. Ők egy nagyon nagy és szokatlan jellemző halmazt használnak, ahol *minden pixelt* egy d dimenziós bináris vektorba képeznek (ha az x értéken $[0, d-1]$ között egy sajátságos pixelt találnak, akkor az $x.$ dimenzió értéke 1 lesz, a többi 0). Minden pixel bináris vektorát konkatenálják egy nd dimenziós egyszerű vektorba (n a pixelek száma). Az osztályozó szabály egy perceptron, amely az input vektor minden dimenziójához rendel egy súlyt. A Winnow tanulási folyamat egy olyan megoldáshoz konvergál, ahol sok ezek közül a súlyok közül nulla. Mindamellett a jellemzők egy igen nagy halmazát megtartja (néhány százat vagy ezret).

2.2.2. Tanítási eredmények

A 2.2.1. definíció egy olyan fogalmat vezet be, amely a detektorok hatékonyságát fogja reprezentálni.

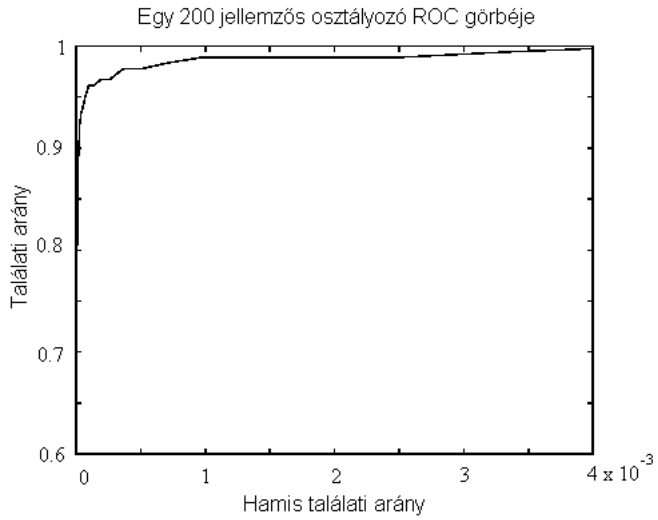
2.2.1. Definíció. ***ROC görbe****: A ROC (Receiver Operating Characteristic) görbe egy olyan valós→valós függvény grafikonja, amely megadja, hogy egy detektornak adott hamis találati arány mellett mekkora a legjobb találati aránya.*



2.4. ábra. A ROC görbe számítása.

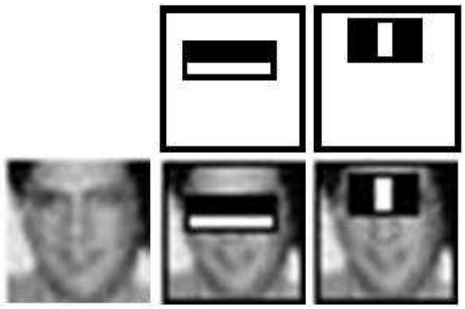
Egy *detektor architektúra* vizsgálata során egy architektúrát többször tesztelünk, különböző paraméterekkel, így több *konfigurációt* kapunk egy architektúrához. A következő leírás a ROC görbe meghatározását adja *egy detektorhoz*, ha azt *több konfigurációban* is teszteljük. Minden egyes konfigurációhoz találati arány – hamis találati arány párosokat kapunk a tesztelés folyamán, melyek megmutatják, hogy bizonyos hamis találat mellett mekkora találati arányt érhetünk el. Ezek az értékpárok pontokként vannak ábrázolva a 2.4. ábrán. Egy hamis találati arányhoz több pont is tartozhat, mivel vannak olyan konfigurációk, melyek azonos hamis találati aránnyal, de különböző találati aránnyal rendelkeznek. Az azonos hamis találati aránnyal rendelkező pontok közül kell kiválasztani a tetőpontot, vagyis azt amelyiknél a legnagyobb a találati arány. Nevezzük ezt a találati arányt a detektor architektúra találati arányának adott hamis találati arány mellett. Így kapjuk az ábrán a szaggatott vonallal összekötött tetőpontok görbéjét. Ebből a detektor architektúra ROC görbéjét a következőképp kapjuk: a tetőpontokra egy monoton növekvő görbét illesztünk, mert feltehetjük, hogy a ROC görbe monoton növekvő. Ugyanis ha egy találati arányt el tudunk érni egy hamis találati arány mellett, akkor feltételezhetjük, hogy egy ennél nagyobb hamis találati arány mellett is el tudjuk azt érni. A 2.4. ábrán folytonos vonal jelzi a tetőpontokra illesztett ROC görbét.

Az arcdetektor tanulásának és a teljesítményének részletei a 7.1. fejezetbe kerültek. Néhány egyszerű eredményről essen szó előzetesen. A kezdeti kísérletek alapján egy 200 jellemzőből alkotott osztályozó már gyakorlatban használható eredményt ad (lásd 2.5. ábra). Adott 95%-os detektálási arány mellett az osztályozó elérte, hogy 1 hamis pozitív eredményt produkált a 14084 darabos teszthalmazon.



2.5. ábra. 200 jellemzős osztályozó ROC görbéje.

Az arcdetektáláshoz az AdaBoost által kiválasztott jellemzők szemléletesek és könnyűszerrel értelmezhetők. Az első kiválasztott jellemző úgy tűnik arra a tulajdonságra koncentrál, hogy a szemkörnyék területe gyakran sötétebb, mint az orr és az arc területe (lásd 2.6. ábra). Ez a tulajdonság relatív nagy összehasonlítva a detektálás során használt alablakokkal, és ezért valamennyire érzéketlen a méretre és az arc helyzetére. A második választott tulajdonság arra épít, hogy a szemek sötétebbek mint az orr vonala.



2.6. ábra. Az AdaBoost által választott első és második jellemző. A két jellemző látható a felső sorban, az alsó sor pedig a jellemzőket egy tipikus tanító arcra helyezve mutatja meg. Az első jellemző az intenzitásbeli különbséget méri a szemterület és a felső arcterület között. A jellemző kihasználja, hogy a szemkörnyék gyakran sötétebb az arcnál. A második jellemző a két szem és az orrnyereg területét hasonlítja össze.

Összegezve a tapasztalatokat elmondható, hogy egy 200 jellemzős osztályozó kezdeti bizonyíték arra, hogy egy osztályozó, melyet téglalap jellemzőkből alkotunk, hatékony technikát jelent az objektum detektálás terén. A detektálás szempontjából ezek az eredmények jelentősek, de még nem elegendőek valós idejű folyamatokhoz. A számításigény szempontjából ez az osztályozó valószínűleg gyorsabb minden eddig

publikált rendszernél, egy 384x288-as képet 0.7 másodperc alatt néz át. Sajnos a legtöbb technika, ami növeli a detektálás teljesítményét, az osztályozóhoz jellemzőket adva, azonnal növeli a számítási időt is.

2.3. A figyelmi kaszkád

Ez a fejezet osztályozók kaszkádjának alkotására szolgáló algoritmust mutat be, ami növeli a detektálási arány, ugyanakkor a számítási időt jelentősen csökkenti. Az alapötlet, hogy könnyű olyan gyenge osztályozókat készíteni, melyek elutasítják a hamis alablakok többségét, míg majdnem minden pozitív példát detektálnak. Egyszerű osztályozókat használunk tehát a negatív példák elutasítására, és az összetettebb osztályozókat csak a bíztató területeken futtatjuk le, hogy minél alacsonyabb hamis pozitív arányt érjünk el.

A kaszkád szintjei az AdaBoostot használó osztályozók tanulása során jönnek létre. Egy két jellemzős erős osztályozóval kezdve hatékony arcszűrőt kaphatunk az erős osztályozó küszöbének olyan beállításával, amely minimalizálja a hamis negatívok számát. Az AdaBoost kezdeti küszöbértéke, $\frac{1}{2}\sum_{t=1}^{T}\alpha_t$, a tanítóhalmazon minimalizálja a hibaarányt. Egy alacsonyabb küszöb magasabb detektálási arányt és magasabb hamis pozitív arányt eredményez. A mérésekre alapozva egy érvényes tanítóhalmazt használva, a két jellemzős osztályozó beállítható, hogy az arcok 100%-át detektálja 40% hamis pozitív eredmény mellett. Lásd a 2.6. ábrát az ebben az osztályozóban használt két jellemző leírására.

Detektálási teljesítménye a két jellemzős osztályozónak gyakorlatban még nem elfogadható. Mindamellett az osztályozó kevés művelettel jelentősen csökkenti azoknak az alablakoknak a számát, melyek további feldolgozást igényelnek:

1. A téglalap jellemzők kiértékelése (jellemzőnként 6-9 tömbhivatkozás).

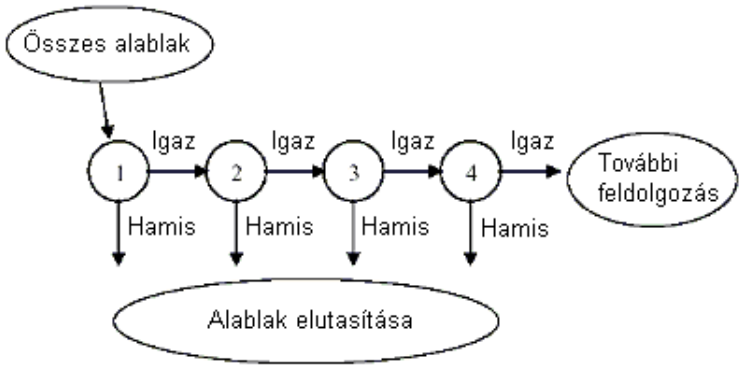
2. A gyenge osztályozó kiszámítása (minden jellemzőre egy műveletet, egy küszöbölést igényel).

3. A gyenge osztályozók kombinálása (jellemzőnként egy szorzást, egy összeadást és végül egy küszöbölést igényel).

A két jellemzős osztályozó körülbelül 60 mikroprocesszor utasításba kerül. Nehéz elképzelni, hogy bármilyen egyszerűbb szűrővel magasabb elutasítási arányt érhetünk el. Összehasonlítva egy egyszerű kép letapogatással, vagy egy egyrétegű perceptronnal, 20-szor kevesebb időt vesz igénybe a két jellemzős módszer alablakonként.

Az általános formája a detektálási folyamatnak egy degenerált döntési fa, melyet kaszkádnak nevezünk [18] (lásd 2.7. ábra). Egy pozitív eredmény az első osztályozóból kiváltja a második osztályozó kiértékelését, amely szintén úgy van beállítva, hogy magas detektálási arányt érjen el. Egy pozitív eredmény a második osztályozóból kiváltja a harmadik osztályozó kiértékelését, és így tovább. Egy negatív eredmény bármely szinten az alablak azonnali elutasításához vezet.

A kaszkád struktúra azt a tényt tükrözi, hogy egy kép túlnyomó többségű alablaka negatív. Ezért a kaszkád minél több negatívot próbál elutasítani a legkorábbi



2.7. ábra. A detektor kaszkádjának sematikus ábrázolása. Osztályozók sorozatán megy keresztül minden alablak. A kezdeti osztályozó a negatív példák nagy részét elutasítja nagyon kevés számolással. A szekvencia további szintjein további negatív példák esnek ki, de itt már több számítás árán. Néhány szint után az alablakok száma jelentősen lecsökken. További számításokat is alkalmazhatunk, növelhetjük a szintek számát (mint a most leírt detektorban is) vagy hozzáadhatunk egy alternatív detektor rendszert is.

szinteken. Mivel egy pozitív egyed kiváltja az összes szint kiértékelését, ez egy rendkívül ritka esemény.

Hasonlóan egy döntési fához, a részsorozat egy osztályozóját azokkal a példákkal képezzük ki, melyek az összes előző szakaszon átmentek. Ennek eredményeként a második osztályozó sokkal bonyolultabb mint az első. A példák melyek átmennek az első szinten, „nehezebbek" mint a tipikus példák. A még bonyolultabb példák a mélyebb osztályozóknál lefelé nyomják a ROC görbét. Adott detektálási arány mellett, a mélyebben lévő osztályozók hamis pozitív aránya megfelelően magasabb.

2.3.1. Osztályozók kaszkádjának tanítása

A kaszkád előállító folyamat minél magasabb detektálási arányt és minél alacsonyabb detektálási műveletigényt próbál meg elérni. Az arcdetektálás terén régebbi rendszerek jó eredményt értek el (85 és 95% közötti detektálási arányt) és nagyon alacsony hamis pozitív arányt (10^{-5} és 10^{-6} közöttit). A kaszkád szintjeinek száma és a szintek mérete elegendő nagy kell legyen, hogy legalább hasonló detektálási teljesítményt érjünk el, miközben a számításigényt csökkentjük.

Tegyük fel, hogy adott az osztályozók kaszkádja, melynek hamis pozitív aránya legyen

$$F = \prod_{i=1}^{K} f_i,$$

ahol F a kaszkád hamis pozitív aránya, K az osztályozók száma, és f_i az i. osztályozó hamis pozitív aránya a példákon, melyek eljutnak hozzá. A detektálási arány

$$D = \prod_{i=1}^{K} d_i,$$

ahol D a kaszkád detektálási aránya, K az osztályozók száma, és d_i az i. osztályozó detektálási aránya a példákon, melyek eljutnak hozzá.

Meghatározhatjuk a cél arányokat a kaszkád folyamat minden szintjéhez, hogy adott hamis pozitív arányt és detektálási arányt elérjünk. Például elérhetünk 0.9 detektálási arányt egy 10 szintes osztályozóval, ha minden szint detektálási aránya 0.99, mivel $0.9 \approx 0.99^{10}$. Míg ennek a detektálási aránynak az elérése ijesztően hangozhat, addig jelentősen könnyít a dolgon, hogy minden szintnek csak körülbelül 30%-os hamis pozitív arányt kell elérnie ($0.30^{10} \approx 6 \times 10^{-6}$).

A kép pásztázása során kiértékelt jellemzők száma szükségszerűen egy valószínűségi folyamat függvénye. Bármely alablak addig halad a kaszkádon, egyszerre csak egy osztályozón át, míg el nem döntjük, hogy az ablak negatív, vagy ritka körülmények között az alablak minden teszten végigmegy, így pozitív eredményt ad. A folyamat várt viselkedését meghatározza a kép ablakok szétosztása egy tipikus teszt halmazon. Minden osztályozó kulcsmértéke a "pozitív aránya", azoknak az ablakoknak az aránya, melyekre az osztályozó pozitív választ ad. A kiértékelt jellemzők számának várható értéke:

$$N = n_0 + \sum_{i=1}^{K} \left(n_i \prod_{j<i} p_j \right)$$

ahol N a kiértékelt jellemzők várható száma, K az osztályozók száma, p_i az i. osztályozó pozitív aránya, és n_i a jellemzők száma az i. osztályban. Érdekes módon, attól fogva, hogy az objektumok nagyon ritkák, a "pozitív arány" egyenlő a hamis pozitív aránnyal.

A kaszkád elemeit tanító folyamat némi odafigyelést igényel. Az AdaBoost tanulási folyamatról szó volt a 2.2. fejezetben. Az AdaBoost a hibák minimalizálására törekszik, és nem kifejezetten magas detektálási arányok elérésére tervezték magas hamis pozitív arányok költsége mellett. Egy általános példa, a hibák elkerülésére az AdaBoost által létrehozott perceptron küszöbének beállítása. Magas küszöb kevesebb hamis pozitívat detektáló osztályozót eredményez, alacsonyabb detektálási aránnyal. Alacsony küszöbbel pedig több hamis pozitívat és magasabb detektálási arányt elérő osztályozót kapunk.

A teljes tanítási folyamat kétféle, egymásnak ellentmondó kritérium optimalizálását igényli. Többnyire egy osztályozó több jellemzővel magasabb detektálási arányt és alacsonyabb hamis pozitív arányt fog elérni. Ugyanakkor egy több jellemzővel rendelkező osztályozó több számítási időt igényel. Alapjában véve definiálhatunk egy optimalizációs keretet, amiben

- az osztályozó szintek száma,
- a jellemzők száma, n_i, minden szinthez, és
- minden szint küszöbe

úgy kivitelezett, hogy minimalizáljuk N jellemzők számát, adott F és D célértékek mellett. Sajnos ennek a minimumnak a megtalálása egy igen nehéz feladat.

Gyakorlatilag egy nagyon egyszerű alkalmazást használnak hatékony osztályozók előállításához. A felhasználó kiválasztja a minimum elfogadható f_i és d_i arányokat.

A kaszkád minden szintjét AdaBoosttal tanítják, hogy a jellemzők számát addig növelik, míg a célul kitűzött detektálási és hamis pozitív arányokat el nem érik. Az arányokat a detektor tesztelésével kapják meg egy érvényes halmazon. Ha még nem érték el a célul kitűzött hamis pozitív arányt, további szinteket adnak a detektorhoz. A negatív halmazt a következő szintek tanításához az aktuális detektor futtatása során kapott hamis pozitív eredményekből kapják. Az algoritmus pontosabban:

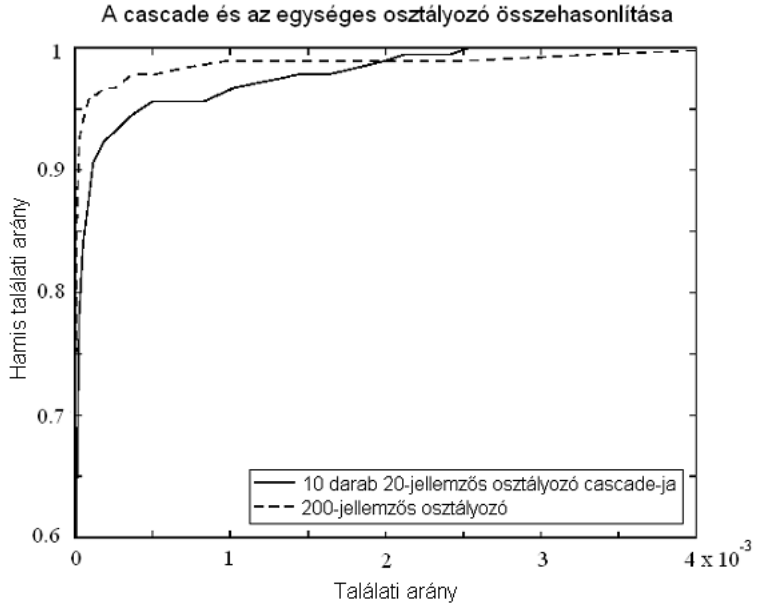
- Határozzuk meg az f értéket, a maximálisan elfogadható hamis pozitív arányt, és d értéket, a minimálisan elfogadható találati arányt a kaszkád minden szintjéhez.
- Határozzuk meg a teljes kaszkádra vonatkozó hamis pozitív arányt, F_{target}-et.
- P = a pozitív példák halmaza.
- N = a negatív példák halmaza.
- $F_0 = 1.0; D_0 = 1.0$
- $i = 0$
- amíg $F_i > F_{target}$
  - $i \leftarrow i + 1$
  - $n_i = 0; F_i = F_{i-1}$
    - $n_i \leftarrow n_i + 1$
    - Használjuk P és N halmazokat egy n_i jellemzős osztályozó tanítására az AdaBoostot algoritmussal.
    - Értékeljük ki az aktuális kaszkád osztályozót az érvényességi halmazon az F_i és D_i meghatározásához.
    - Csökkentsük az i. osztályozó küszöbét, amíg az aktuális kaszkád osztályozó eléri a minimum $d \times D_{i-1}$ detektálási arányt (ez befolyásolja F_i értékét is).
  - $N \leftarrow \emptyset$
  - Ha $F_i > F_{target}$, akkor értékeljük ki az aktuális kaszkád detektort a nem-objektum (azaz nem-arc) képeken és tegyünk minden hamis detektálást az N halmazba.

2.3.2. Egy egyszerű kísérlet

Azért, hogy megmutassák a kaszkád megközelítés előnyeit, két egyszerű detektort tanítottak: egy monolitikus 200-jellemzős osztályozót és egy kaszkádot 10 darab 20-jellemzős osztályozóval. A kaszkád első szintje 5000 arc és 10000 nem-arc alablakon tanult a nem-arc képekből véletlenszerűen választva. A második szint ugyanazon az 5000 arcon tanult és az első osztályozó 5000 hamis pozitív eredményén. Ez a folyamat folytatódott, hogy a szekvencia minden következő szintje is az előtte levő szint hamis pozitív eredményeivel tanult.

Az egységes 200-jellemzős osztályozó az előbb leírt kaszkád szintjeinek tanítása során használt példák unióján tanult. Jegyezzük meg, hogy a kaszkád osztályozó nélkül nehéz lett volna kiválasztani a nem-arc tanító példákat a monolitikus osztályozó tanításához. Használhattuk volna a nem-arc képek összes lehetséges alablakát, ez azonban ésszerűtlenül hosszúvá tette volna a tanulási időt. A kaszkád osztályozó a tanulás során hatékonyan csökkenti a nem-arc tanítóhalmazt úgy, hogy a könnyű példákat kidobja, és a nehéz példákra fókuszál.

A 2.8. ábra mutatja a két osztályozó összehasonlításának eredményét. Látható, hogy pontosságban kevés eltérést mutatnak, ugyanakkor a sebességben nagy a különbség (az ábra erre nem tér ki). A kaszkád osztályozó közel 10-szer gyorsabb, mert már az első szinten kidobja a nem-arcok többségét, így azokat a szekvencia többi szintje nem értékeli ki.



2.8. ábra. A 200-jellemzős osztályozó és az egységes osztályozó összehasonlító ROC-görbéi. A pontosságban nincs jelentős különbség, viszont a kaszkád majdnem 10-szer gyorsabb, mint a monolitikus osztályozó.

2.3.3. A detektor kaszkád diszkussziója

Az elképzelés hasonló a Rowley és társai által leírt rendszerhez [3]. Rowlay és társai két neuronhálót tanítottak. Az egyik háló meglehetősen komplex volt, mely a kép kis régiójára fókuszált, és alacsony hamis pozitív aránnyal detektált arcot. Egy másik, sokkal gyorsabb hálót is tanítottak, mely a kép nagyobb területére fókuszált, és magas hamis pozitív aránnyal detektált arcot. Rowley és társai ez utóbbi gyorsabb hálót a kép elővizsgálatára használták azért, hogy a pontosabb de lassabb háló számára megtalálják a jelölt régiókat. Habár nehéz pontosan meghatározni, úgy tűnik, hogy az általuk fejlesztett két neuronhálós rendszer a leggyorsabb létező arcdetektor. A Viola-Jones rendszer egy hasonló megközelítést használ, de a két szintet 32

szintre terjeszti ki.

A kaszkád detektáló folyamat struktúrája lényegében egy degenerált döntési fa, és mint ilyen kapcsolatban áll Amit és German munkájával [6]. Eltérően az egyetlen fix detektort használó rendszerektől, Amit és German egy alternatív álláspontot javasolnak, ahol egyszerű képjellemzők szokatlan közös előfordulásait használják egy összetettebb detektáló folyamat kiértékelésének előidézéséhez. Így nem szükséges a teljes detektálási folyamatot kiértékelni minden potenciális skálán és pozícióban. Az ő implementációjukban először néhány jellemző-detektort kell kiértékelni minden pozícióban. Azután ezeket a jellemzőket csoportosítják, hogy a szokatlan együttes előfordulásokat megtalálják. Gyakorlatilag mióta a Viola-Jones detektor alakja és a jellemzők, melyeket használ extrém hatékonyak, a detektor minden skálán és pozícióban történő kiértékelésének amortizációs költsége sokkal kisebb, mint megtalálni és csoportosítani az éleket a képen.

2.4. Detektálás

2.4.1. Képfeldolgozás

Minden példa alablak szórásnégyzetét normalizálták a tanításhoz, hogy a különböző megvilágítások hatását csökkentsék. A detektálás során ezért szintén szükség van normalizációra. Egy kép alablakának szórásnégyzete gyorsan kiszámítható egy integrális kép páros használatával. Emlékeztetőül $\sigma^2 = m^2 - \frac{1}{N}\Sigma x^2$, ahol σ a standard eloszlás, m a középérték, és x az alablak egy pixelének intenzitásértéke. Egy alablak középértéke az integrális kép használatával kiszámítható. A pixelek négyzetének összege a kép négyzetének integrális képe alapján kiszámítható (pl. két integrális képet használunk a letapogató folyamatban).

2.4.2. Detektor letapogatási folyamat

A végső detektor a képet többszörös skálán és pozícióban tapogatja le. A skálázást elérhetjük a detektor skálázásával a kép skálázása helyett. A folyamat érthető, mert a jellemzők minden skálán ugyanakkora költséggel számíthatók. Jó eredményt kaphatunk 1.25 körüli skálázási (vagy növekedési) faktorral.

A detektort a következőképpen használjuk: az első letapogatás után néhány képponttal (Δ-val) eltoljuk a letapogató ablakot, így kapjuk a következő pozíciót. Az eltolás mértéke a skálázástól függ, ha az aktuális skála s, akkor az ablakot $[s\Delta]$-val toljuk el, ahol [ ] az egész értéket jelöli.

A választott Δ mind a detektor sebességére mind pedig a pontosságra hatással van. Mivel a tanító képeknek van néhány eltolási változója, a tanított detektor jó detektálási eredményt ér el a kis eltolások ellenére. Ugyanis a detektor alablak több mint egy pixellel eltolható egy alkalommal. Habár több mint egy pixelnyi eltolással kissé csökkentjük a detektálási arányt, ugyanakkor csökkentjük a hamis pozitívak számát is.

2.4.3. Többszörös detektálások integrálása

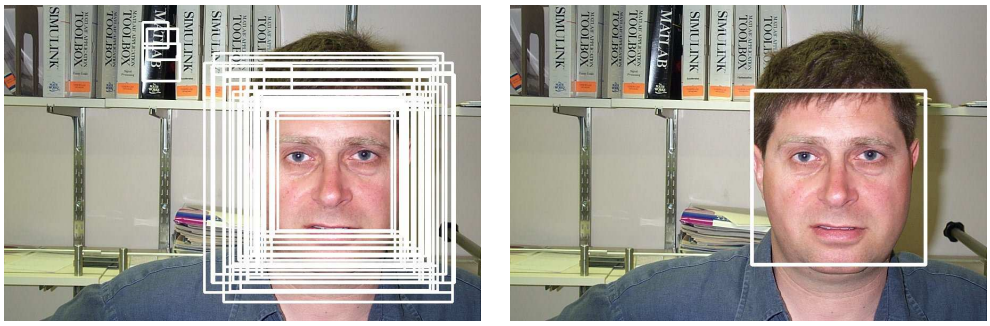
Mivel a végső detektor érzéketlen az eltolás és a skálázás kis változásaira, többszörös detektálás fordulhat elő minden arc körül a letapogatott képen. Ez gyakran előfordul hamis pozitív eredményeknél is. Gyakorlatban egy archoz egy detektálást szeretnénk. Ennek eléréséhez hasznos lehet az alablakok egy utófeldolgozását elvégezni, mely az egymást fedő detektálásokat egy detektálássá kombinálja össze.

A következő módszer alapján a detektálásokat nagyon egyszerűen kombinálták össze. A detektálásokat először diszjunkt halmazokba osztjuk. Két detektálás ugyanabba a halmazba kerül, ha azok fedik egymást. Minden felosztás egy egyszerű végső detektálást ad. A sarkai a végső határoló területnek a halmazba tartozó detektálások sarkainak átlaga.

Néhány esetben ez az utófeldolgozás csökkenti a hamis pozitív eredmények számát, mivel az egymást fedő hamis pozitív eredmények egy detektálássá integrálódnak.

2.4.1. Definíció. ***Topográfiai heurisztika, topográfiai csoportszám****: Egy detektor egy objektumot többféle ablakmérettel is képes észrevenni. Ekkor ezek a detektálások egy csoportot alkotnak. A topográfiai heurisztika a következőképpen működik: Egy adott elemszámnál nagyobb csoportból egy "átlag" területet készít. Ezek a területek kerülnek be a detektor eredménylistájába. Ezt az elemszámot* topográfiai csoportszámnak *nevezzük. A topográfiai csoportszámnál kevesebb, vagy ugyanolyan elemszámú csoportokat eldobja a heurisztika.*

A 2.9. képen az arcdetektort láthatjuk működés közben, heurisztika nélkül és heurisztikával, azaz 0 és 3 topográfiai csoportszámmal.



(a) **(b)**

2.9. ábra. Arcdetektor heurisztika nélkül és 3-as topográfiai heurisztikával

3. fejezet

Szemdetektorok tanítása

Viola és Jones az Intel Open Computer Vision könyvtárban[1] közzétették arcdetektorukat. Ezt az arcdetektort, melynek eredményeit a 7.1. fejezet mutatja be, felhasználhatjuk C alapú nyelvben írt programjainkban. A könyvtár tartalmazza az objektum detektáló rendszerük tanító algoritmusát és tesztelő függvényét is. Ennek segítségével pedig bárki taníthat saját detektorokat, csak megfelelő mennyiségű pozitív és negatív példaképet kell hozzá adnia, és megfelelően kell beállítania a tanítófüggvény számos paraméterét. A paraméterek megfelelő beállítása nehézkes folyamat. Egy detektor tanítása akár egy hétig is eltarthat, de mivel a detektorok szintekből állnak, melyek szöveges fájlként tároldónak, ezért a tesztelést már a korai tanulási fázisban elkezdhetjük, és ha nem találjuk elég eredményesnek a még tanuló detektort egy másikkal szemben, akkor le is állíthatjuk a tanítást. A detektor kaszkád egy szintje egy erős osztályozó függvénynek felel meg (2.3. fejezet).

A tanító függvénnyel szemdetektorokat tanítottam: szemkörnyék, balszem és jobbszem detektorokat. A következőkben a tanításról és a kapott detektorok összehasonlításáról lesz szó. A tanítás során kezdettől fogva teszteltem az eredményeket, így azok alapján döntöttem, hogy leállítsak-e egy tanuló detektort, és hogy miben módosítsam a következőként indított tanítás paramétereit. Párhuzamosan több mint 10 detektort tanítottam. Az összes detektorból tizet hasonlítok össze az alábbiakban, 2 szemkörnyék detektort, 5 balszem detektort és 3 jobbszem detektort.

A tanítás során a következő paraméterek befolyásolták döntően a detektorok eredményét, így ezekre fogok részletesebben kitérni az összehasonlításokban:

- Pozitív és negatív példák mennyisége.
- Tanítópéldák mérete.
- Szimmetria beállítása.

A bal és jobb szemeket nem lett volna szükséges megkülönböztetni, ha feltesszük, hogy azok többnyire szimmetrikusak. Néhány esetben azonban egyáltalán nem szimmetrikus a szem. Még ekkor sem szükséges külön balszem és jobbszem detektorokat használnunk, mert a képből kivágott területet, ahol nagy eséllyel balszem szerepel, tükrözhetjük függőlegesen, majd detektálhatjuk a jobbszem detektorral. Jobbszemet hasonlóan detektálhatunk balszem detektorral. Mégis tanítottam balszem és

[1] `http://sourceforge.net/projects/opencvlibrary/`

jobbszem detektorokat is, az összehasonlítás kedvéért. Ezek közül vannak olyan párok, melyek ugyanazokon a képeken tanultak, csak a tanító képek egymás tükrözöttjei. Ezek között a detektorpárok között is találunk teljesítménybeli eltérést.

3.1. Képadatbázisok

3.1.1. Arc adatbázisok

A *tanításhoz* Feret arc adatbázist[2] használtam. Az adatbázis összesen 14051 fekete-fehér képet tartalmaz, ebből 3880 frontális arckép (láss néhány példát a 3.1. ábrán), melyek különböző méretűek, többnyire 256x384 pixel felbontásúak. A képekhez metaadatok is tartoznak, melyek tartalmazzák a bal- és jobbszemek középpontjának a képhez viszonyított relatív koordinátáit. Az adatok alapján kivágtam a képekből a megfelelő részeket a tanításhoz.

(a) **(b)** **(c)**

3.1. ábra. Frontális arcok a Feret arc adatbázisból.

A *teszteléshez* egy 450 frontális arcból álló arc adatbázist[3] használtam (láss néhány példát a 3.2. ábrán), melyek 27 különböző emberről készültek. A képek színesek, 896x592-es felbontásban. Az arc adatbázis körülbelül 10 darab a detektálás szempontjából hátrányos képet tartalmaz. Van amelyiken a szemét eltakarja a kép alanya, vannak rajzolt arcok, és vannak túlságosan sötét képek a tíz között. Ezeket nem vettem ki az adatbázisból a tesztelés során. Ehhez az adatbázishoz nem voltak metaadatok, ezeket én készítettem el hozzá.

3.1.2. Negatív példák

A tanításhoz negatív példákat is meg kell adnunk. Ehhez a legjobb ha sokféle képünk van, melyek egyáltalán nem ábrázolják a tanítandó objektumot. Több különböző adatbázist is használtam egyszerre (az internet címek 2005. júniusában elérhetőek voltak):

[2] `http://www.itl.nist.gov/iad/humanid/feret/`
[3] `http://www.vision.caltech.edu/Image_Datasets/faces/faces.tar`

(a) (b) (c)

3.2. ábra. A teszteléshez használt arcadatbázis.

1. 450 fekete-fehér kép, melyek témája teljesen különböző, 378x251-es felbontásban.
   `http://www.vision.caltech.edu/Image_Datasets/background/background.tar`

2. Falevelekről készített fotók gyűjteménye, 186 színes képet tartalmaz, 896x592-es felbontásban.
   `http://www.vision.caltech.edu/Image_Datasets/leaves/leaves.tar`

3. Egy szoba belseje különböző szögekből, 116 képet tartalmaz, 1024x768-as felbontásban.
   `http://www-2.cs.cmu.edu/~owenc/word/cluttera.htm`

4. Egy saját gyűjtemény természeti képekből: 102 színes kép, változó felbontásban.
   `http://www.ephotograph.com/`

3.2. A detektorok

3.2.1. Szemkörnyék detektorok

eyes2-30x15-3547-450-sym vs. eyes2-40x20-3547-450-sym

(Az elnevezés a következőkre utal: tanító adatfile[4] neve: eyes2 – a tanító képek mérete: 30x15 pixel ill. 40x20 pixel – pozitív példák száma: 3547 darab – negatív példák száma: 450 darab – a detektor szimmetriája: igaz)

Két szemkörnyék detektort tanítottam, ezek összehasonlítása következik. Az egyik detektort 30x15 pixel felbontású példaképekkel tanítottam, a másikat 40x20 pixel felbontásúakkal. Arra számítottam, hogy a nagyobb képek jobb detektort eredményeznek majd, hiszen több információt tartalmaznak. Az eredmények azonban mást mutatnak. Minden más beállítás megegyezett a két detektorban: 3547 pozitív példán és 450 negatív képen tanultak, és a tanított objektumot, azaz a szemkörnyéket, szimmetrikusként állítottam be.

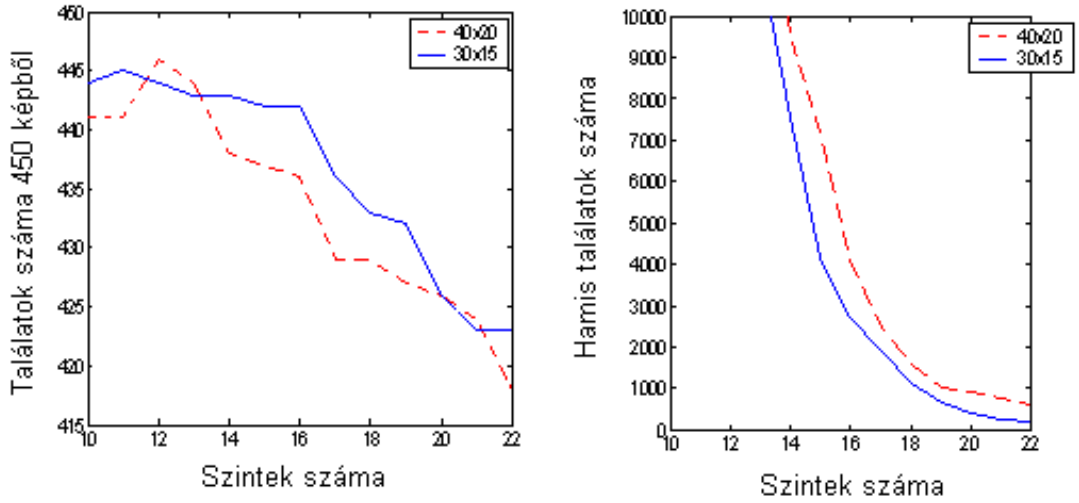
A pozitív példák a Feret arcadatbázisból származnak, melyben minden képhez tartozik egy metaadatfájl, ami megmondja a bal és a jobb szemgolyók középpontjának koordinátáit a képen belül. Ez alapján kivágtam a szemkörnyékeket a kö-

[4]Ezt az adatfile-t a pozitív képekből a tanítás előtt kell elkészíteni.

vetkezőképpen: Jelöljük d-vel a két szemgolyó pixelben mért távolságát. Ekkor a kivágott szemkörnyék bal felső koordinátái a képen belül, ha (x, y) a kép baloldalán lévő szemgolyó koordinátái: $(x - 0.5 * d, y - 0.5 * d)$. A kivágott rész jobb alsó koordinátái: $(x + 0.5 * d, y + 0.5 * d)$. Így a kivágott rész $2 * d$ széles, és d magas lett. A kivágott szemkörnyékekre láthatunk néhány példát a 3.3. ábrán. A tanításhoz a kivágott képekből az egyik detektor számára 30x15-ös méretű képeket generáltattam a tanító programmal, a másik detektor számára pedig 40x20 pixel méretűt. Negatív példaként az 1. számú negatív példákat tartalmazó adatbázist használtam.

3.3. ábra. Az arcokból kivágott szemkörnyékek, melyeket a tanításhoz használtunk.

A 30x15-ös méretű képeket felhasználó detektor gyorsabban tanult a 40x20-as méretűnél. Ebben semmi meglepő nincs, figyelembe véve, hogy egy 30x15-ös méretű képben kevesebb az információ, mint egy 40x20-as méretűben, hiszen ugyanabból a képből generálom mindkettőt, csak az egyiket kisebbre tömörítem a másiknál. Ami viszont meglepő, hogy mégis a kevesebb információt felhasználó detektor mutat jobb eredményeket (lásd a Függelékben 7.2. táblázatot). Tehát nem csak gyorsabban tanult a másiknál, de még jobban is detektál. A detektorokat összehasonlíthatjuk a 3.4. ábrán.



(a) **(b)**

3.4. ábra. Szemkörnyék detektorok összehasonlítása: (a) Találatok száma 450 képből, (b) hamis találatok száma.

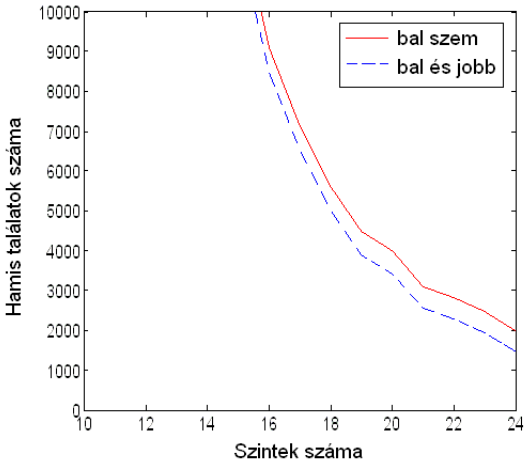
3.2.2. Balszem detektorok

A balszem és jobbszem detektorokat háromféleképpen teszteltem, egyrészt azért, hogy megtaláljam a megfelelő paramétereket a tanításokhoz, másrészt hogy a gya-

korlati alkalmazás körülményeit vizsgálhassam.

Az első tesztelési módszerben a képen szereplő kísérleti személy balszeme volt az egyetlen megtalálandó objektum, és a detektor a teljes képet vizsgálta. A detektorok első tesztelési módszerének eredményét mutatják a 7.2-7.6. táblázatok.

A második módszerben a balszem és a jobbszem egyaránt helyes találatnak számított, mivel a két szem gyakran nem mutat jelentősebb eltérést. A 3.5. ábra mutatja, az első és második tesztelési módszer közötti különbséget. Az első módszer szigorúbb, így ebben az esetben a detektorok több hamis találatot mutatnak, melyek közül sok tulajdonképpen a jobbszem. Ezért ha elfogadjuk a második módszer alapján a jobb szemeket is helyes találatként, akkor mint azt az ábra is mutatja, kevesebb hamis találatunk lesz.



3.5. ábra. A tesztelési módszerek összehasonlítása. Az első esetben csak a balszem találata számít helyesnek, míg a második esetben a jobb szemet is elfogadjuk helyes találatnak a két szem hasonlósága miatt.

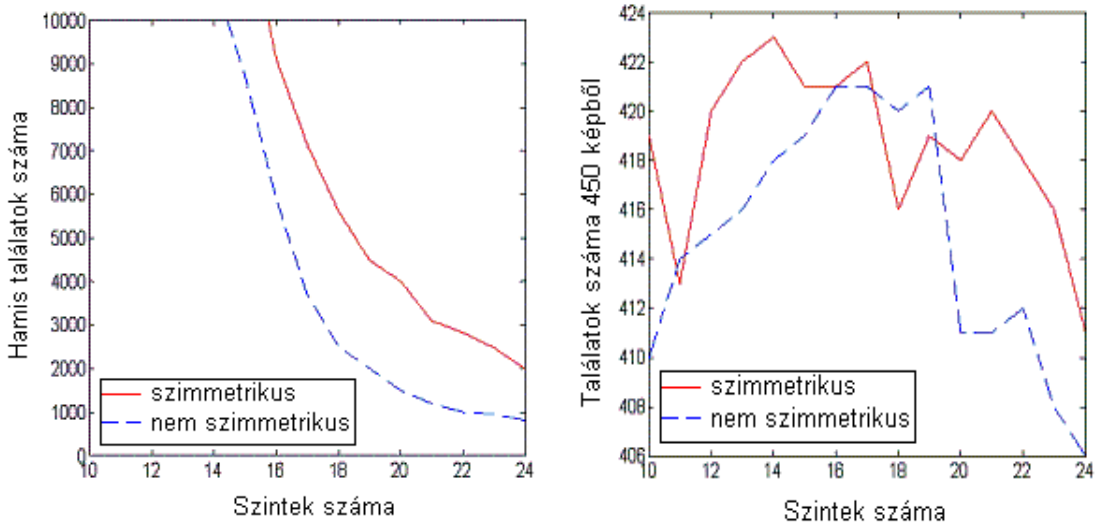
A harmadik módszer a hierarchikus tesztelés volt. A detektorokat gyakorlat során egy szemkövető programban használtam fel (4. fejezet). Hierarchikus rendszert építettem belőlük. Elsőként arcdetektort futtattam a kameraképen, az arcdetektor eredményén belül szemkörnyék detektort, majd a szemkörnyék detektor eredményén belül balszem és jobbszem detektorokat. A gyakorlati felhasználás indokolta a hierarchikus tesztek készítését is. A tesztelésben a detektorok azonban nem egymás eredményeivel dolgoztak, hanem a tesztadatbázisból egzakt módszer alapján kivágott képeken. A szemkörnyék detektorok a kivágott arcokon futottak, míg a balszem és jobbszem detektorok a kivágott szemkörnyéken. A hierarchikus tesztelés táblázatos eredményeit mutatják a 7.7-7.11. táblázatok.

left2-25x20-3545-450-sym vs. left2-25x20-3545-450-nonsym

Az első kettő balszem detektor 3545 darab 25x20 pixel méretű tanító képen (lásd 3.6. ábrát) és 450 negatív példán tanult. A kettő közötti különbség a szimmetria beállításában volt. Mivel a szem a minták nagy többségében szimmetriát mutat, érdemes azt szimmetrikusként kezelni. Ugyanakkor elképzelhetőnek tartottam, hogy

3.6. ábra. A balszem detektorok tanításához használt 25x20 pixel méretű példaképek.

mégis a nem szimmetrikus detektorok fognak jobb eredményeket mutatni. A sejtés félig helyesnek bizonyult az első tesztelési módszer alapján. A szimmetrikus detektor több helyes találatot adott, ugyanakkor a nem szimmetrikus detektor kevesebb hamisat. A két detektor összehasonlítását a 3.7. ábra mutatja.



(a) **(b)**

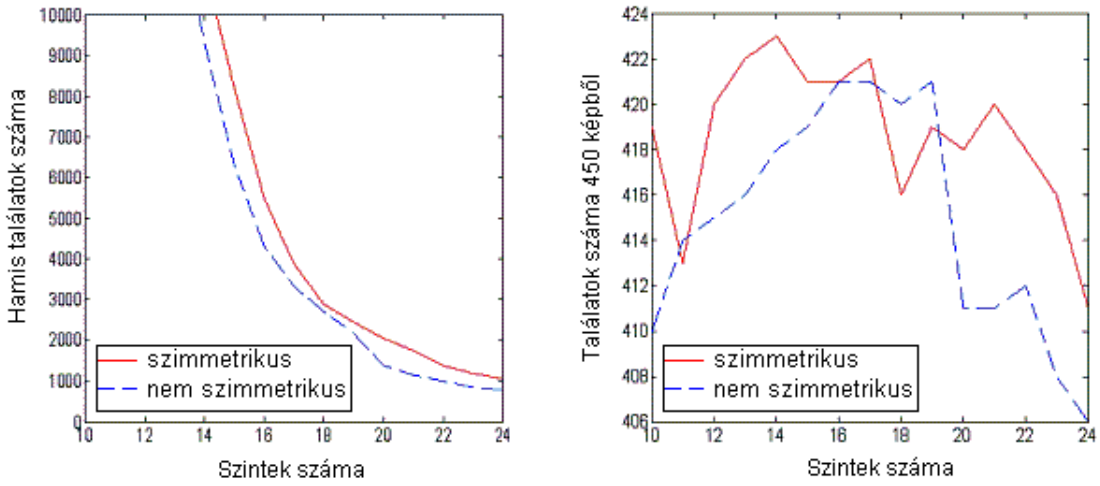
3.7. ábra. A szimmetrikus és a nemszimmetrikus left2-25x20-3545-450 balszem detektorok összehasonlítása. A szimmetrikus balszem detektor több helyestalálatot ad a 450 képet tartalmazó tesztadatbázison (b), viszont a nemszimmetrikus detektor kevesebb hamis találatot mutat (a).

left3-25x15-7097-450-sym és left3-25x15-7097-450-nonsym

Azért, hogy több tanítópéldám legyen, felhasználtam az arcképeken mindkét szemet. A jobbszemeket vertikálisan tükröztem, így balszemeket kaptam. Ezen az úton kétszer annyi tanítópéldám lett, mint eredetileg. A 3.8. ábrán látható néhány tanítópélda.

Ezek a detektorok a tanítópéldák méretében és mennyiségében térnek el az előző két detektortól. A tanítópéldákhoz szűkebben vágtam ki az arcképekből a szemeket. A kivágást vertikálisan szűkítettem. Habár ezzel információt veszíthet a detektor, a szemkörnyék detektorokhoz hasonlóan nyertem is a szűkítéssel, csökkent a hamis detektálások száma. Ezt mutatja a 3.9. ábra.

3.8. ábra. A balszem detektorok tanításához használt 25x15 pixel méretű példaképek. A tanításhoz az arcképek mindkét szemét felhasználtam, a jobbszemeket vertikálisan tükröztem. A képpárok közül tehát az egyik a kivágott balszem, a másik pedig a tükrözött jobbszem.

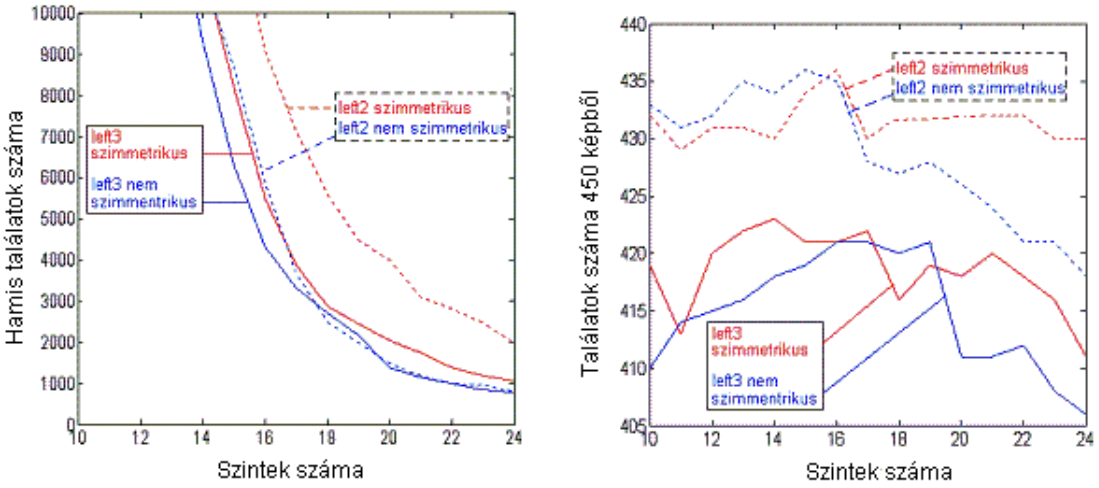


3.9. ábra. A szimmetrikus és a nemszimmetrikus left3-25x15-7097-450 balszem detektorok összehasonlítása a tanítóképek szűkítése és azok számának növelése mellett. A szimmetrikus balszem detektor több helyestalálatot ad a 450 képet tartalmazó tesztadatbázison, viszont a nemszimmetrikus detektor kevesebb hamis találatot mutat.

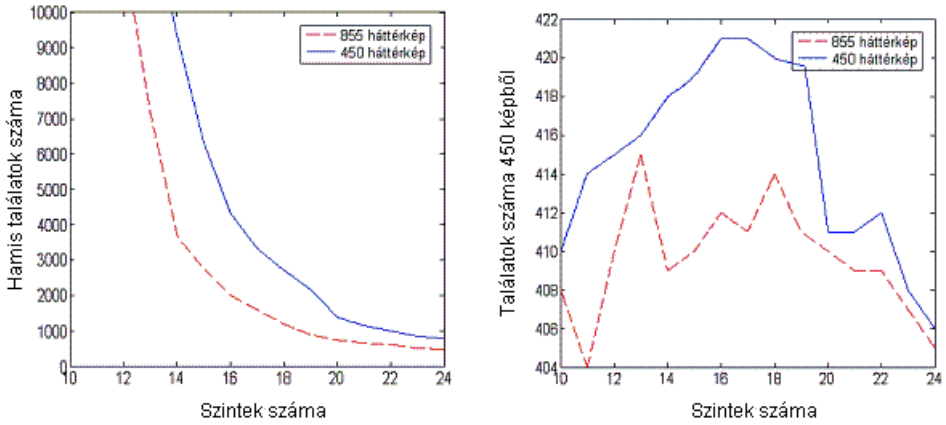
A tesztelési eredmények mutatják, hogy a kisebb tanítópéldákkal készített detektorok valamivel rosszabb eredményeket mutatnak a detektálás terén, viszont kevesebb hamis találatot érnek el. Ezt mutatják a 3.10. ábra grafikonjai.

left3-25x15-7097-855-nonsym

A következő detektorban a negatív példák számát növeltem. Ez is nemszimmetrikus detektor volt, 25x15-ös méretű tanítópéldákkal, melyek között a tükrözött jobbszemek is szerepeltek. Ehhez a detektorhoz a negatív példák 1-4. adatbázisokat használtam. A 3.11. ábra mutatja ennek a detektornak és a left3-25x15-7097-450-nonsym detektornak az összehasonlítását. A két detektor csak a negatív példák számában tért el egymástól. A több negatív példán tanuló detektor kevesebb hamis detektálást mutat, a kevesebb negatív példán tanuló pedig több helyes detektálást. Mivel a 24. szint környékén a két detektor szinte ugyanakkora detektálási arányt mutat, ezért gyakorlatban a több negatív példán tanuló detektort használtam fel.



3.10. ábra. A left2-25x20-3545-450 és a left3-25x15-7097-450 detektorok összehasonlítása szimmetrikus és nemszimmetrikus esetben egyaránt. A kisebb tanítópéldákkal készített left3-25x15-7097-450 detektorok egy-két százalékkal rosszabb eredményeket mutatnak a detektálás terén, viszont kevesebb hamis találatot érnek el.



3.11. ábra. A left3-25x15-7097-450-nonsym és a left3-25x15-7097-855-nonsym detektorok összehasonlítása. A több negatív példán (háttérképen) tanuló detektor kevesebb hamis detektálást mutat, a kevesebb negatív példán (háttérképen) tanuló pedig több helyes detektálást. A 24. szint környékén a két detektor megközelítőleg ugyanakkora detektálási arányt mutat.

Balszem detektorok összefoglalás

Öt balszem detektort tanítottam. Különbség a szimmetria beállításában, a pozitív példák méretében, számában, és a negatív példák számában volt. Az eltérésekkel az egyik oldalon veszteség ért: csökkent a detektálási arány, míg a másik oldalon nyereséges volt a változtatás: csökkent a hamis találati arány. Ilyen változtatás volt a szimmetria hamisra állítása, a pozitív példák méretének csökkentése, számuk növelése, és a negatív példák számának növelése. A Függelékben 7.3. és 7.4. táblázatok mutatják az egyes detektorok teszteredményeit az első módszer alapján, azaz csak

a balszemet keresve.

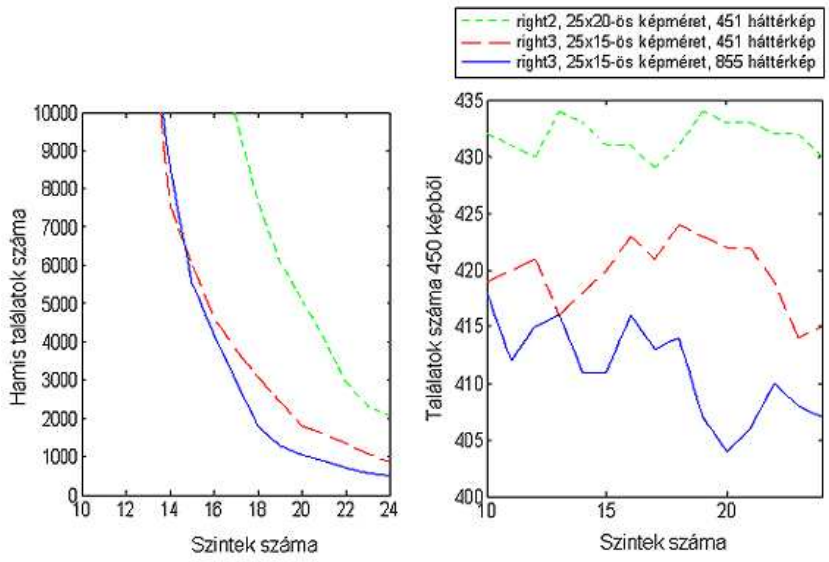
3.2.3. Jobbszem detektorok

Három jobbszem detektort tanítottam, hasonló beállításokkal, mint a balszemdetektoroknál:

1. right2-25x20-3563-450-nonsym,
2. right3-25x15-7041-450-nonsym,
3. right3-25x15-7041-855-nonsym.

Az első detektor 25x20 pixel felbontású tanítópéldákat használt fel, míg a következő kettő 25x15 pixel felbontásút. Az első detektor az arcképekből csak a jobb szemeket használta fel, míg a következő kettő a függőlegesen tükrözött balszemeket is. Az első két detektor 450 negatív példán tanult, míg a harmadik detektor 855 negatív példán. Az eredmények ugyanazt mutatják mint a balszem detektorok esetében, azaz kisebb felbontású tanítóképek mellett, azok számának növelésével, és a negatív példák számának növelésével a detektálás aránya csökken, és a hamis találatok aránya szintén csökken.

A 3.12. ábra mutatja a jobbszem detektorok összehasonlító grafikonjait, a Függelékben 7.5. és a 7.6. táblázatok pedig a teszteredményeket.



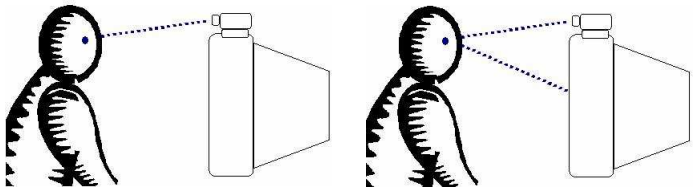
3.12. ábra. A jobbszem detektorok összehasonlítása. Kisebb felbontású tanítóképek mellett, azok számának növelésével, és a negatív példák (háttérképek) számának növelésével a detektálás aránya csökken, és a hamis találatok aránya szintén csökken.

4. fejezet

Szemkövető program

4.1. Szemkövetés

Az ember-számítógép interakció egyik fontos eszköze a tekintet követés, aminek feltétele a robusztus szemkövetés (a kettő közötti különbséget a 4.1. ábra mutatja). A szemkövetés során a szemgolyó mozgását figyeljük, a tekintet követés során pedig a fixációs pontot. Tekintet alatt egy ember aktuális látósugarát értem. A fixációs pont a látósugár és a megfigyelt tárgy, például a képernyő felszínének metszéspontja. A tekintet irányát a szemkövetés eredményeiből határozhatjuk meg. A tekintetet használhatjuk a felhasználó szándékának értelmezéséhez. A szemmozgás felhasználása az ember és a számítógép együttműködésében számos előnyt jelent. Például a felhasználó tekintetének meghatározása segíthet az utasítások értelmezésében és képessé teheti a számítógépet, hogy megállapítsa a felhasználó kognitív állapotának néhány jellemzőjét, ilyenek például a zavarodottság és a fáradtság.



4.1. ábra. (a) Szemkövetés és (b) tekintetéskövetés

A tekintet iránya megmutatja a felhasználó érdeklődésének tárgyát és betekintést nyújt az éppen zajló kognitív folyamatokba. A tekintet valós idejű megfigyelése lehetőséget ad olyan változások megmutatására, melyek a szemmozgás térbeli és időbeli tulajdonságaitól függnek. Például a tekintet alapján meghatározhatjuk a felhasználó fixációs pontját a képernyőn, amiből következtethetünk, milyen információ érdekli éppen a felhasználót. Ez megfelelő eseményeket idézhet elő, mint például a megfigyelt terület felbontásának növelése zoomolással. Egy másik példa a sávszélesség beosztása úgy, hogy csak az az információ jelenik meg az internetes böngészőben magas felbontással, amit a felhasználó éppen néz. A tekintet követés felhasználható mozgáskorlátozott emberek kommunikációjában, a beviteli hatékonyság fokozásában, katonai alkalmazásokban, játékokban, mobil eszközökben.

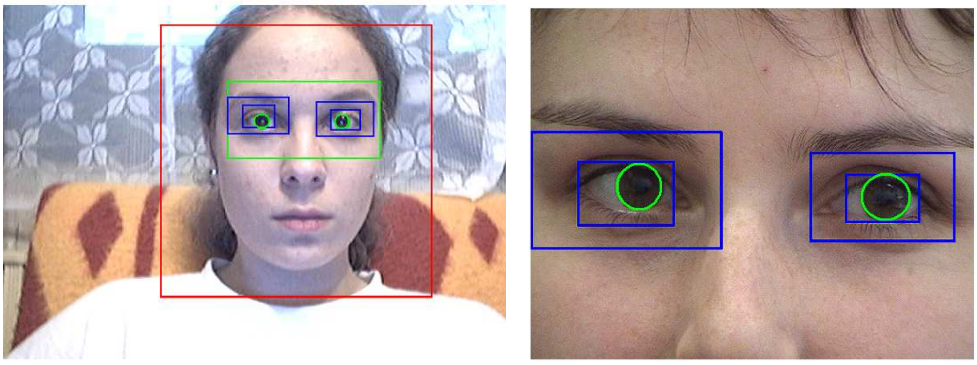
A szemkövetéshez speciális kamerákat használnak, melyeknek nagy a felbontása, kicsi a látószöge, esetleg monitorba építhetők, nagy sebességűek, gyakran speciális megvilágítást igényelnek, például infravörös fényt, és pontosak, ugyanakkor drágák. Vannak fejre szerelhető speciális eszközök is, melyek például egy szemüvegbe építhetők. A speciális infra rendszernél a fényforrás helyzete ismert, így a szemgolyón létrejövő csillanások egymáshoz viszonyított helyzetéből meghatározható a tekintet iránya.

4.2. Szemkövetés webkamerával, szemkövető program

A webkamerás szemkövetés egyik nagy előnye, hogy nem igényel drága felszerelést, egy jobb minőségű webkamera ma már pár ezer forintért beszerezhető. A szemkövető program, melyet az ELTE NIPG csoportjában fejlesztünk, szemkövetést végez, egészen pontosan szemgolyó követést. A kameraképen megkeresi és követi a felhasználó szemgolyóinak köreit. A program C++ programnyelven Microsoft Windows környezetben készül.

Viola-Jones detektorokat használunk az arc, azon belül a szemkörnyék majd a bal- és jobbszem keresésére (a detektorok elméleti hátterét lásd a 2. fejezetben). A detektorok eredményén belül Hough körkeresés fut, és a szemgolyók időbeli jellemzői alapján szűrjük a szemgolyó jelölteket. A 4.2.a. ábra mutatja a detektálások és a körkeresés eredményét egy 640x480-as webkamera képen, a 4.2.b. ábra pedig egy 720x576-os felbontású, zoomolható kézi kamera képén végzett detektálást mutat. A 4.3. fejezet bemutatja, hogy a Viola-Jones detektorokat milyen hierarchiába szervezve használja fel a szemkövető program.

Jó minőségű webkamera már közel alkalmas a szemmozgás vizsgálatára. A szemgolyó követés hibái tovább javíthatók ha a különböző lehetőségek valószínűségének becslésével validáljuk vagy elvetjük az aktuális választást. Ha a validáció nem sikeres, valószínűleg célszerű a detektorok együttes valószínűségei alapján dönteni, például a szakértők szorzatának (Products Of Experts) elve alapján, ami az együttes valószínűségek súlyozásával ad eredményt.



(a) **(b)**

4.2. ábra. (a) Szemkövetés 640x480 felbontású webkamera képen, (b) szemkövetés 720x576 felbontású zoomolható kézikamera képen.

4.3. Detektorok hierarchiája a programon belül

Az arc, szemkörnyék, balszem és jobbszem detektorok egymás eredményeit figyelembe véve a kép egyes területein futnak csak le (lásd példaként a 4.2.a. ábrát). A detektorok téglalapok sorozatával térnek vissza, melyek a keresett objektumot tartalmazhatják. Míg az arcdetektor a teljes képen lefut, addig a szemkörnyék detektor csak az arcdetektor legnagyobb eredményét dolgozza fel, amennyiben az arcdetektor talált eredményt. Ha az arcdetektor nem talált eredményt, vagy ha az arcdetektálást kikapcsoljuk, mert például zoomolható kamerát használunk (4.2.b. ábra), mellyel a szemkörnyékre közelítünk, akkor a szemkörnyék detektor a teljes képen fog lefutni. Ha tehát arcdetektálást végzünk először, és kapunk is eredményt, akkor a szemkörnyék detektor az eredménylista legnagyobb, legutolsó eredményén belül fog lefutni, ellenkező esetben a teljes képen. A szemkörnyék detektor eredménylistájából már nem a legutolsó és egyben legnagyobb eredményét választom ki a további bal- és jobbszem kereséshez, hanem azok közül az eredmények közül választom a legnagyobbat, melyekre teljesülnek az alábbi feltételek:

- A szemkörnyék detektor által talált téglalap szélessége nagyobb legyen a legnagyobb arcdetektor eredmény szélességének 50%-ánál.
- A szemkörnyék detektor által talált téglalap a legnagyobb arcdetektor eredmény felső felében legyen, figyelembe véve, hogy egy emberi arcon a szemek az arc felső felében helyezkednek el.

Ha van ilyen eredménye a szemkörnyék detektornak, akkor a balszem és jobbszem detektorokat azon belül futtatom, ha nincs akkor az arcdetektor eredményén belül, ha annak sem volt eredménye, akkor pedig a teljes képen. Szintén a teljes képen futtatom a balszem és jobbszem detektorokat, ha nem használunk arc és szemkörnyék detektorokat. Ha viszont használunk szemkörnyék detektort, akkor a jobbszem detektor a fenti szempontok alapján kiválasztott szemkörnyék megfelelő jobb felén fut, a balszem detektor pedig a bal felén. Ha az egyik oldali detektor nem talál eredményt, akkor a képrészletet vertikálisan tükrözzük, és a másik oldali detektort futtatjuk rajta.

Felmerülhet a kérdés, miért szükséges külön balszem és jobbszem detektor használata, egy egységes szemdetektor használata helyett. A detektorok tanítása során (lásd 3. fejezetet) arra az eredményre jutottam, hogy feltételezve, hogy a szem szimmetrikus, több helyes eredményt adnak a tanított detektorok, ugyanakkor több hamis eredményt is, azaz ál-objektumot találnak. Ez abból adódhat, hogy a szem nem minden esetben tökéletesen szimmetrikus, a két szemsarok eltéréseket mutat. Ezért nem szimmetrikus detektorokat használok.

A szemgolyókeresés a Viola-Jones detektorok eredményén belül fut, amennyiben azok találtak keresett objektumot. Amennyiben találtunk bal- és jobbszemet, a szemgolyókeresés ezeken belül fut. Tapasztalati mérések alapján a balszem és jobbszem detektorok eredményét továbbszűkíthetjük (4.2.b. ábra). Csak azokat a köröket fogadjuk el lehetséges jelöltként, melyeknek a középpontja az így szűkített területekbe esik. Ez a szűkítés negyed szélességet von le balról és jobbról, ötöd magasságot pedig felülről és alulról a balszem és a jobbszem detektorok eredményéből.

Amennyiben nem találunk bal- és jobbszemet, a körkeresés a szemkörnyéken belül fut, ha azt sem találtunk, akkor az arcon belül, végül pedig, ha arcot sem találtunk, a teljes képen. A keresési területek ugyanígy változnak, ha direkt kikapcsoljuk az arckeresést, vagy az arc és szemkörnyék keresést, illetve az arc, szemkörnyék, balszem, jobbszem keresést.

A szemkövető program teljes detektorhierarchiáját futtatva minden beérkező képen 10 kép/másodperc teljesítményt érhetünk el, 640x480 pixel felbontású webkamera képen egy 1.6 GHz-es Intel Pentium M processzorral rendelkező notebook-on. A program megfelelő futásához azonban elég, ha csak minden ötödik képen futtatjuk a program teljes detektor hierarchiáját, így pedig 20 kép/másodperc teljesítményt érhetünk el.

4.4. Kör és körrészelet detektálás

A webkamera képen történő szemgolyó követés megvalósításban fontos, hogy mi jellemzi a szemgolyót, milyen módszerek alkalmasak ezeknek a jellemzőknek megtalálására, milyen bemenet kell ezek felismeréséhez (felbontás, képminőség, színek, sebesség, stb.), milyen számításigénye van a módszernek és mennyibe fog kerülni.

A szemgolyó tulajdonképpen kör, mely környezeténél sötétebb. A körök felismerésére Hough transzformációt alkalmazhatunk, mely elemi geometriai alakzatok felismerésére igen jó módszer, viszonylag gyors és megbízható (lásd a 7.3. fejezetet).

A szemgolyónak megfelelő körök időbeli viselkedése jellemző, ebből következően a „hamis" körök gyorsabban kiesnek a jelöltek közül. Több jelölt fenntartása mellett, nem pillanatnyi jósággal dolgozunk, hanem ahhoz, hogy egy kör magas jóságot (minőséget) érjen el, elvárható tőle, hogy sok képen keresztül legyen jelen.

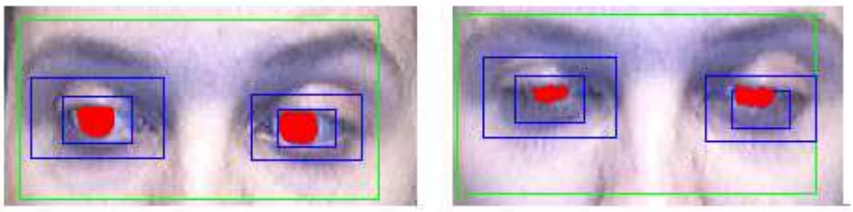
A szemgolyó követés egyik problémája, hogy pislogás során elveszítjük a szemgolyó köröket. A szemkövető program teljes detektorhierarchiáját futtatva minden beérkező képen 10 kép/másodperc teljesítményt érhetünk el, 640x480 pixel felbontású kameraképen egy 1.6 GHz-es Intel Pentium M processzorral rendelkező notebook-on. A program megfelelő futásához azonban elég, ha csak minden ötödik képen futtatjuk a program teljes detektor hierarchiáját, így pedig 20 kép/másodperc teljesítményt érhetünk el. Szemgolyó keresést minden képen végzünk. Pislogás során a szemgolyó körök legalább 5 képről el fognak tűnni. Mivel a körök időbeli viselkedését figyeljük, a problémából előnyt is kovácsolhatunk. Az 5 kép alatt a program még nem veszíti el a szemgolyók körülbeli helyét, ugyanakkor egy nagyobb ugrás vehető észre a körök középponjának helyzetében. Az ugrás alapján körülbelül 80%-ban észrevehető, ha a felhasználó pislog, ehhez pedig utasítást köthetünk.

A szemgolyó követés másik problémája, hogy a felhasználók egy részénél a szemgolyóból nem látszik a teljes kör, helyette csak egy „csíkot" látunk. Mosolygás során pedig a legtöbb ember szemgolyójából kevesebb látszik, mint egy szenvtelen arckifejezés mellett. Ennek a hibának a javítására megpróbálhatunk kör helyett körrészletet keresni a Hough transzformációval. Egy másik lehetőség az összefüggő komponensek keresése.

Az összefüggő komponens keresést a balszem és a jobbszem detektorok eredményén belül futtatom. Összefüggő komponensek keresése során először küszöbölést végzünk a képrészeleten, így egy bináris képet kapunk, melyen összefüggő fekete il-

letve fehér részeket találunk majd. A küszöb helyes beállítása kulcsfontosságú, mint ahogy a Hough transzformációnál is. Ezután a legnagyobb fekete összefüggő rész lesz a szemgolyó. Elképzelhető, hogy a folyamat nagyon világos szemek esetén nem ad jó eredményt, mert a szemszín a bőrszínnel hasonló telítettségű.

A módszer hátránya, hogy csak abban az esetben érdemes összefüggű komponenst keresnünk, ha a balszem és a jobbszem detektoroknak van eredménye. A detektorok képesek leszűkíteni annyira a kereső területet, hogy biztosan a szemgolyó legyen a legnagyobb összefüggő komponens. Ezért ez a megoldás önmagában nem teszi lehetővé a folyamatos szemkövetést, ugyanakkor a Hough transzformációt kiegészítheti. Az összefüggő komponens keresésre mutat példát a 4.3. ábra.



4.3. ábra. Összefüggő komponensek keresése a balszem és a jobbszem detektorok eredményén belül. A pirossal színezett rész mutatja a legnagyobb összefüggő komponenst.

4.5. A program tesztelése gyerekfilmeken

A szemkövető programot a Bliss alapítvány Segítő Kommunikációs Központjában készített filmeken teszteltem. Hét mozgáskorlátozott és/vagy szellemi sérült fiatallal készült együttesen mintegy 3 órás anyag állt rendelkezésemre. A felvételek tömörített, 720x576 képpont méretű, 25 kép/másodperc sebességű videó fájlok voltak, melyeket zoomolható kézi kamerával vettek fel. Az ELTE NIPG csoportjában készült fejegér[1] használatának tanulása folyamán készültek a felvételek.

Mivel mozgássérült fiatalokról van szó, akik a fejegér használatát tanulták, a fejmozgás jelentős, ami csökkenti a feldolgozási sebességet. A fényviszonyok szempontjából megfelelő, azaz világos felvételeken, ahol a kamera beállítás is idális, tehát a felhasználó feje végig benn van a kameraképben, nem lóg ki sem az álla, sem a fej oldalsó része, a detektor hierarchia futásakor a következő idő eredményeket kaptam: A hierarchia futtatása minden képen 9 kép/másodpercre csökkentette a feldolgozási időt. Minden képen futtatni a teljes detektor rendszert erőforrás pazarlás. Ha minden hetedik képen futtattam, akkor 16 kép/másodperc teljesítményt kaptam, és ennél jobb eredményt már nem tudtam elérni. A szemkövető program beállítása szerint körkeresés minden képen fut.

Az arc, szemkörnyék, balszem és jobbszem detektorok a fej oldalra döntésében kb. 30 fokot még gond nélkül elviselnek, a legjobb ebből a szempontból a fejdetektor, amely akár 45 fokban oldalra döntött fejet is képes detektálni, de alacsonyabb találati

[1] http://nipg.inf.elte.hu/headmouse/headmouse.html

aránnyal. A fej oldalra fordításakor szintén kb. 30 fok a detektorok tűrési határa. A körkeresés annál jobban és gyorsabban dolgozik, minél szűkebb keresési területet kap a detektorok alapján. Ha a detektorok közül egyik sem talál eredményt, például mert a felhasználó nem tudja egyenesen tartani a fejét, így gyakran nagy fokban oldalra dönti, a körkeresés még ekkor is megtalálja többnyire az egyik szemet. Mindkettőt csak ritka esetben. A körkeresés leggyakrabban a következő hibákat ejti: szemgolyó helyett orrlyukat talál, vagy a szemöldökre, szájra, fülre téved, ezek kontúrjai erősen láthatók az éldetektált képen.

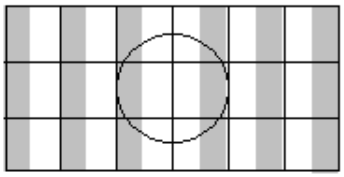
Az arcdetektor legrosszabbul azokon a felvételeken dolgozik, ahol az áll vonala, vagy az arc széle nem látszik. Ezek az információk meghatározóak az arcdetektorban, így ha lemaradnak, nem ismeri fel az arcot.

4.6. Irány utasítás neuronhálókkal

A szemkövető program jelenlegi verziójában felhasználó függő módszerrel képes irány utasítások generálására. Az irány utasítás generálása nyíl-billentyű események szimulálását jelenti. A felhasználó függőség alatt azt értem, hogy mindenkinek, aki szeretné használni a programnak ezt a funkcióját, egy fáradtságos tanítási folyamatban kell résztvenni, ez körülbelül 30 percet jelent. (Néhány ingyenesen felhasználható beszédfelismerő rendszerben, hasonlóan be kell tanítanunk a programot, hogy az felismerje a kiejtett szavainkat.)

A nyíl-billentyű események akkor generálódnak, ha a felhasználó balra, jobbra, fel vagy lefele néz, a képernyő megfelelő széleire. Ha középre néz a felhasználó, nem generálódik kimenet, így ezt tekinthetjük a nyugalmi állapotnak. A nyíl-billentyű eseményekkel például játékokat vezérelhetünk. Ha nem generálunk semmilyen billentyű eseményt, akkor is hasznos lehet az az információ, hogy a felhasználó a monitornak éppen melyik felére néz. Ha látható, hogy már régóta a képernyő bal felén dolgozik, akkor ráközelíthetünk erre a területre. Szemgesztusokat is definiálhatunk, például gyors jobb-bal-jobb szemmozgáshoz köthetünk valamilyen utasítást.

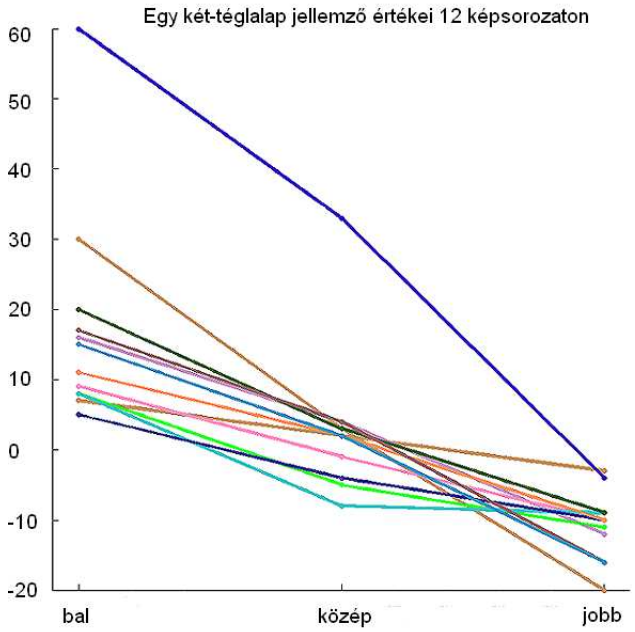
Az irány utasítás megvalósításában a 2.1. fejezetben bemutatott jellemzőket használom fel. Tanulórendszerként pedig mesterséges neuronhálókat használok. A szemgolyó keresés eredménye alapján a szemgolyóhoz viszonyítva 18 darab két-téglalap jellemzőt helyezek fel mindkét szemre (4.4. ábra). Ezekben a két-téglalap jellemzőkben a fehér területeken található képpontok intenzitásainak összegét vonom ki a sötét területen található képpontok intenzitásainak összegéből.



4.4. ábra. A szemgolyóra felhelyezett 18 darab két-téglalap jellemző.

A jelenleg használt két-téglalap jellemzőket kísérletezés alapján választottam ki. Olyan jellemzőkre volt szükség, melyek együttesen öt irány megkülönböztetésére ké-

pesek: amikor a felhasználó középre néz, balra, jobbra, fel és le. A bal-közép-jobb szemgolyó pozíciókat a kipróbált két-téglalap jellemzőknek kb. 50%-a tudta szétválasztani, a fel-közép-le szemgolyó pozíciók szétválasztása azonban nehezebb feladatnak bizonyult. Eleinte receptív mezőket és három-téglalap jellemzőket is kipróbáltam. A receptív mezők tulajdonképpen egy-téglalap jellemzőként értelmezhetők, egy területen belül elhelyezkedő képpontok intenzitásértékeinek összegét nevezem így. A receptív mezők és a három-téglalap jellemzők nem bizonyultak elég hatékonynak a sem a bal-közép-jobb, sem a fel-közép-le szempozíciók szétválasztásában. A tesztelések alatt külön vizsgáltam a következő két hármast: a bal-közép-jobb és a fel-közép-le szempozíciókat. A 4.5. ábra mutatja a egyik jellemző értékeit a bal-közép-jobb szempozíciók alatt, egy 6 emberről készült 2-2 sorozatot tartalmazó tesztadatbázison. Látható, hogy ez a jellemző, ami egyébként két-téglalap jellemző volt, egyértelműen szétválasztja a bal-közép-jobb szempozíciókat mind a 12 képsorozaton. A fel-közép-le szempozíciók ennyire egyértelmű megkülönböztetésére sajnos egyik jellemző sem volt képes. Ez az eredmény valószínűleg abból adódik, hogy a szem igen kis különbséget mutat, ha a felhasználó középre ill. a monitor felső felére néz.



4.5. ábra. Az egyik két-téglalap jellemző értékei a bal-közép-jobb szemgolyó pozíciókban.

Az első neuronhálók 3 jellemző értékeit használták fel. A mesterséges neuronhálók elméleti leírása a 7.4. fejezetbe került. A kezdeti hálók három szempozíció megkülönböztetésére szolgáltak: bal, közép és jobb. Minden pozícióhoz egy hálót tanítottam, melyeknek három rétegük volt 6-6-1 neuron számmal, ahol a 6. neuron a bemeneti és a rejtett rétegben konstans értékek voltak. A backpropagation tanuló algoritmust használtam fel a tanításhoz (az algoritmus leírását lásd a 7.4.2. fejezetben). Aktivációs függvényük szigmoid függvény volt. A két szemre külön tanítottam

a neuronhálókat, tehát a háló együttes 6 neuronhálót tartalmazott (bal-közép-jobb hálókat mindkét szemre) és a párok eredményét logikai *és* függvény alapján kombináltam. Egy emberről készült 10 sorozaton tanultak (egy sorozat most a bal, közép és jobbra néző szempozíciójú képeket tartalmazta) és ugyanennek az embernek másik 10 sorozatán tesztelem őket. A tesztelés alapján 93%-os eredményt kaptam az egyik szemen dolgozó hálóval, 90%-os eredményt a másik szemre. Együttes eredményük az *és* kapcsolat alapján 83% volt, közös tévedés pedig nem fordult elő. Ugyanez a hálóegyüttes egy 12 sorozatot tartalmazó adatbázison, mely 6 emberről 2-2 sorozatot tartalmazott, a következő eredményeket adta: első szemre 61%, 2. szemre 80%, együttesen 50%, közös tévedés pedig nem volt. Ezek az eredmények már jelezték, hogy az egy emberről készült képeken tanuló háló nem lesz képes általánosítani, azaz csak azon az egy emberen fog jól működni.

A hálóegyüttes, melyet jelenleg használok, gyökeresen eltér a kezdeti próbálkozásoktól. Mivel a hálónak a fel és a le irányt is meg kell tudnia különböztetni, a 3 jellemző kevésnek bizonyult. A tanításokhoz maximum 30 jellemzőt használtam fel, ez azonban már soknak bizonyult, a jelenleg használt hálók 18 darab két-téglalap jellemzővel dolgoznak. A réteg szám maradt, azaz van egy bemeneti réteg 18 + 1 (konstans) neuronnal, egy rejtett réteg ugyanennyi neuronnal és egy kimeneti réteg 5 neuronnal. Praktikusabb volt a szemekhez egy-egy hálót használni, melyek öt kimeneti neuronja az öt lehetséges szempozíciónak felel meg. Nem csak kényelmesebb volt ezeknek a használata, de jobb eredményt is adtak. Az utolsó hálóegyütteseket már online felhasználásban teszteltem, a szemkövető programba beépítve, nem a számszerű eredményeket vizsgáltam. A felhasználás során látható volt, hogy a hálóegyüttesek mire képesek. Az általam használt hálóegyüttes 30 rólam készült képsorozaton tanult (egy sorozat öt képet tartalmaz bal-közép-jobb-fel-le szempozíciókkal).

Teszteltem a hálók általánosíthatóságát is, azaz tanítottam neuronhálókat több ember képeit tartalmazó kép adatbázison is (10 emberről volt 2-2 képsorozatom). Sajnos ezek a hálók nem értek el magas pontosságot, elképzelhető, hogy kevés tanító példát kaptak, de az is elképzelhető, hogy az általánosítás a pontosság rovására megy.

A neuronhálók használatának több előnye is van. Egyrészt a hálók futtatása a program sebességét nem csökkenti mérvadóan, másrészt a fejmozgásra érzéketlen a rendszer. A FANN[2] (Fast Artifical Neural Networks) könyvtárat használtam fel a szemkövető programhoz. A könyvtár többrétegű előrecsatolt neuronhálók tanítását és használatát teszi lehetővé. Több tanító algoritmus implementációját is tartalmazza, én a backpropagation algoritmust választottam a háló tanításhoz (az algoritmus leírását lásd a 7.4.2. fejezetben).

[2] http://fann.sourceforge.net/

5. fejezet

Összefoglalás

A 2. fejezetben megmutattam az objektum detektálás egy megközelítését, amely alacsony számítási idővel magas detektálási arányt képes elérni. Mindezt három fontos tulajdonságán keresztül láthattuk.

Az első fontos tulajdonság a jellemzők számításához használt integrális kép reprezentáció. Ahhoz, hogy skála invarianciát érjünk el, a detektor rendszernek működnie kell többszörös skálán. Az objektum detektorok többsége a skála invarianciát a kép skálázásával, a képpiramis kiszámításával éri el. A Viola-Jones objektum detektálás ehelyett a detektorokat skálázza. Az integrális kép kiküszöböli a képpiramis több skálán történő kiszámítását, így jelentősen csökkenti a kezdeti képfeldolgozást. Az arcdetektálás területén ez az előny drasztikusan nagy. Az integrális képet használva, az arcdetektálási folyamat előbb ér véget, mint a képpiramis kiszámítása.

A második fontos tulajdonság az AdaBoost-on alapuló jellemző kiválasztás. Ez egy agresszív és hatékony technika a jellemző kiválasztásra. Ennek birtokában a rendszertervező szabadon definiálhatja a jellemzők nagy és komplex halmazát a tanulási folyamat bemeneteként. A kapott osztályozó mindamellett számításigény szempontjából hatékony lesz, mivel a jellemzőknek csak egy kis halmazát kell futási időben kiértékelni. Gyakran a kapott osztályozó egészen egyszerű; a komplex jellemzők nagy halmazában valószínűleg megtalálható az a kevés kritikus jellemző, amely megragadja az osztályozási probléma struktúráját.

A harmadik fontos tulajdonság az osztályozók kaszkádja, mely radikálisan csökkenti a számítási időt, míg növeli a detektálási pontosságot. A kaszkád korai szintjeit úgy konstruáljuk, hogy azok a képek nagy többségét visszautasítsák, így szentelve nagyobb figyelmet az ígéretes területeknek a következő szinteken. A kaszkád struktúrája ugyanakkor egyszerű és homogén.

A 3. fejezet az arcdetektor mintájára tanított szemdetektorokról szólt. Viola és Jones az Intel Open CV könyvtárban közzétették az implementált tanulóalgoritmust, ezt felhasználva szemkörnyék, balszem és jobbszem detektorokat tanítottam. A detektorok tanítása megmutatta, hogy a detektálási és a hamis találati arányt csökkenthetjük a szimmetria hamisra állításával, a pozitív példák méretének csökkentésével, számuk növelésével, és a negatív példák számának növelésével.

A 4. fejezet a szemkövetés és az ezen alapuló tekintet követés lehetőségeit mutatta meg. A tekintet követés az ember-számítógép interakció fontos területe. Jelenleg speciális eszközöket, például infrakamerát igényel egy pontos tekintet követő

rendszer. Ezek az eszközök azonban nem megfizethetők hétköznapi használatra. Az ELTE NIPG csoportjában fejlesztés alatt álló szemkövető program webkamera kép alapján szemkövetést végez, és felhasználó függő módon irány utasítások adására képes. Távlati célunk, hogy tekintet követő rendszert építsünk webkamera kép felhasználásával.

A szemkövető program Viola-Jones detektorokat használ a szemgolyók keresési területének szűkítésére. Detektor hierarchiájában arc, szemkörnyék, balszem és jobbszem detektorokat találunk, melyek ebben a sorrendben futnak, egymás eredményeire támaszkodva. A szemgolyó köröket Hough transzformációval keressük. Ha a felhasználónak nem látszik a teljes szemgolyója, szükség lehet körrészlet keresésre. Erre szintén alkalmazhatjuk a Hough transzformációt, vagy kereshetünk összefüggő komponenseket.

Az irány utasításokhoz két mesterséges neuronhálót használ fel a rendszer, egyet-egyet a bal- és a jobbszemhez. A két háló 18 darab két-téglalap jellemzőt kap bemenetként, a jellemzőket a detektált szemgolyó körök alapján helyezem el a képen. A neuronhálók együttes eredménye határozza meg, hogy generálunk-e irány utasítást. Irány utasítással vezérelhetünk például játékokat, melyek nyíl-billentyű eseményeket fogadnak. Önmagában is hasznos lehet az az információ, hogy a felhasználó a képernyő melyik részét nézi, hogy könnyítsük a munkáját, például ráközelíthetünk erre a területre.

6. fejezet

Köszönetnyilvánítás

Szeretnék köszönetet mondani Lőrincz Andrásnak a témakörben nyújtott gondos útmutatásért, és hogy lehetőséget kaptam az ELTE NIPG csoport tagjaként megismerni a csapatmunka előnyeit rendkívüli emberek között. Köszönöm Hévízi Györgynek türelmét, gyakorlati ötleteit és tanácsait, melyek segítettek mind az elméleti témakör gyorsabb megértéseben, mind a hatékonyabb szoftverfejlesztésben.

Ábrák jegyzéke

Táblázatok jegyzéke

Irodalomjegyzék

[1] Paul Viola and Michael J. Jones. Robust real-time object detection. Technical Report CRL-2001-01, Cambridge Research Laboratory, 2001. `http://www.hpl.hp.com/techreports/Compaq-DEC/CRL-2001-1.pdf`, 2005.

[2] K. Sung and T. Poggion. Example-based learning for view-based face detection. In *IEEE Patt. Anal. Mach. Intell.*, volume 20, pages 39–51, 1998.

[3] H. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. In *IEEE Patt. Anal. Mach. Intell.*, volume 20, pages 22—38, 1998.

[4] H. Schneiderman and T. Kanade. A statistical method for 3d object detection applied to faces and cars. In *International Conference on Computer Vision*, 2000.

[5] D. Roth, M. Yang, and N. Ahuja. A snowbased face detector. In *Neural Information Processing 12*, 2000.

[6] Y. Amit, D. Geman, and K. Wilder. Joint induction of shape features and tree classifiers. 1997.

[7] C. Papageorgiou, M. Oren, and T. Poggio. A general framework for object detection. In *In International Conference on Computer Vision*, 1998.

[8] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of online learning and an application to boosting. In *Computational Learning Theory: Eurocolt '95*, pages 23–37. Springer-Verlag, 1995.

[9] K. Tieu and P. Viola. Boosting image retrieval. In *International Conference on Computer Vision*, 2000.

[10] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. *Ann. Stat.*, 26(5):1651–1686, 1998.

[11] Edgar Osuna, Robert Freund, and Federico Girosi. Training support vector machines: an application to face detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1997.

[12] J.K. Tsotsos, S.M. Culhane, W.Y.K. Wai, Y.H. Lai, N. Davis, and F. Nuflo. Modeling visual-attention via selective tuning. *Artificial Intelligence Journal*, 78(1-2):507–545, October 1995.

[13] L. Itti, C. Koch, and E. Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Patt. Anal. Mach. Intell.*, 20(11):1254–1259, November 1998.

[14] Patrice Y. Simard, Lon Bottou, Patrick Haffner, and Yann Le Cun. Boxlets: a fast convolution algorithm for signal processing and neural networks. In M. Kearns, S. Solla, and D. Cohn, editors, *Advances in Neural Information Processing Systems*, volume 11, pages 571–577, 1999.

[15] William T. Freeman and Edward H. Adelson. The design and use of steerable filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(9):891–906, 1991.

[16] H. Greenspan, S. Belongie, R. Gooodman, P. Perona, S. Rakshit, and C. Anderson. Overcomplete steerable pyramid filters and rotation invariance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1994.

[17] Andrew Webb. *Statistical Pattern Recognition.* Oxford University Press, New York, 1999.

[18] J. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[19] Rudolf K. Bock. Hough transform. Web page, 1998. `http://rkb.home.cern.ch/rkb/AN16pp/node122.html`, 2005.

[20] Hough transform. Web page. `http://www.netnam.vn/unescocourse/computervision/62.htm`, 2005.

[21] D. H. Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern Recognition*, 12(2):111–122, 1981.

[22] C. Kimme, D. H. Ballard, and J. Sklansky. Finding circles by an array of accumulators. *Communications of the Association for Computer Machinery*, 18:120–122, 1975.

[23] P. V. C. Hough. *Method and means for recognizing complex patterns.* 3069654. U. S. Patent, 1962.

[24] Mose Iadorola. Hough transform. A Java applet demonstration. `http://www.vision.ee.ethz.ch/~jhug/MoseIadarola/Hough5.html`, 2005.

[25] Mark A. Schulze. Circular hough transform. A Java applet demonstration, 1996. `http://www.markschulze.net/java/hough`, 2005.

[26] R.C. Gonzales and R.E. Woods. *Digital Image Processing.* Addison-Wesley, 1992.

[27] K. R. Castleman. *Digital Image Processing.* Prentice Hall, 1996.

[28] Anil K. Jain. *Fundamentals of Digital Image Processing.* Prentice Hall, 1989.

[29] William K. Pratt. *Digital Image Processing.* Wiley, 1991.

[30] M. H. Hassoun. *Fundamentals of Artifcial Neural Networks.* The MIT Press, 1995. `http://neuron.eng.wayne.edu/synapse2/tpage3.html`, 2005.

[31] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to The Theory of Neural Computing.* Addison-Wesley, New York, 1991.

[32] J. A. Anderson. *An Introduction to Neural Networks.* The MIT Press, 1995.

[33] A. Tettamanzi and M. Tomassini. *Soft Computing.* Springer-Verlag, 2001.

[34] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proc. of the IEEE Intl. Conf. on Neural Networks*, pages 586–591, San Francisco, CA., 1993.

[35] S. E. Fahlman. Faster-learning variations on backpropagation: An emperical study. 1988.

[36] G. Horváth, editor. *Neurális hálózatok és müszaki alkalmazásai.* Műegyetemi Kiadó, Budapest, 1998.

[37] A. Lőrincz. *Learning Systems I: Artifical Neural Networks.* 1999. `http://people.inf.elte.hu/lorincz/scripts/Eloadas/ANN_Word_v_0.9.zip`, 2005.

[38] J. Levendovszky. `http://www.hit.bme.hu/people/levendov/neuralis`, 2005.

[39] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[40] M. Minsky and S. Papert. *Perceptrons.* The MIT Press, Cambridge, MA, 1969.

[41] R. Schalkoff. *Artifical Neural Networks.* McGraw-Hill, New York, 1997.

7. fejezet

Függelék

7.1. Arcdetektor eredményei

7.1.1. Tanító adathalmaz

Az arcdetektorhoz használt tanítóhalmaz 4916 arcból állt, melyeket 24x24 pixel felbontásra méreteztek. Az arcokat a world wide web-en talált képekből vonták ki. Néhány tipikus példát mutat a 7.1. ábra. Ezek a példák a fejből többet tartalmaztak, mint a Rowley et al. [3] által vagy a Sung [2] által használtak. A kezdeti kísérletek során 16x16 pixel méretű tanítóképeket használtak, ahol az arcokat szorosabban vágták ki, de az eredmény így valamivel rosszabb lett. Feltételezhetően a 24x24-es képek plusz vizuális információkat tartalmaznak a kisebb képekhez képest, mint például az áll, az arc és a haj vonala, melyek segítik a pontosság növelését. A jellemzők természetéből adódóan, a nagyobb méretű alablakok nem lassítják a teljesítményt. Tulajdonképpen a nagyobb alablak által hozzáadott információt használta fel a detektor a nem-arcok korai visszautasítására.

7.1.2. Az arcdetektor kaszkád struktúrája

A végső arcdetektor egy 32 szintes kaszkád, mely összesen 4297 jellemzőt használ.

Az első osztályozó a kaszkádban két jellemzőt használ és a nem-arcok körülbelül 60%-át utasítja vissza, míg azok közel 100%-t detektálja helyesen. A következő osztályozó öt jellemzőt használ és a nem-arcok 80%-át utasítja vissza, míg az arcok majdnem 100%-át detektálja. A következő három szint 20-jellemzős osztályozók, melyeket két 50-jellemzős osztályozó követ, majd öt 100-jellemzős, végül húsz 200-jellemzős. A jellemzők számának sajátságos megválasztása kísérletezések alapján adódott, melyek során a jellemzők számát addig növelték, míg a hamis detektálási arány jelentősen nem csökkent. Több szintet adtak hozzá, míg a hamis pozitív arány az érvényességi halmazon közel nullára nem csökkent úgy, hogy a magas detektálási arányt ugyanakkor megtartotta. A szintek végső száma és a szintek mérete nem változtatta kritikusan a végső rendszer teljesítményét.

A 2-, 5- és az első 20-jellemzős osztályozó 4916 arcon és 10000 nem-arc alablakon tanult (24x24-es méretű példákon) az AdaBoost tanulási folyamat alapján, melyet a 2.2. fejezet ír le. A nem-arc alablakokat 9500 arcot nem tartalmazó kép alablaka-

7.1. ábra. Példák az arcdetektor tanításához használt frontális arcok közül.

iból véletlen választással gyűjtötték. A különböző osztályozók a nem-arc alablakok különböző halmazain tanultak, így biztosítva, hogy az osztályozók valamelyest függetlenek legyenek, és ne ugyanazokat a jellemzőket használják.

A későbbi szintek tanításához használt nem-arc példák a részleges kaszkád használata során a nem-arc képek letapogatásából kapott hamis pozitív eredmények. Maximum 6000 ilyen nem-arc alablakokat gyűjtöttek minden szinthez. A 9500 nem-arc képnek megközelítően 350 millió nem-arc alablaka van.

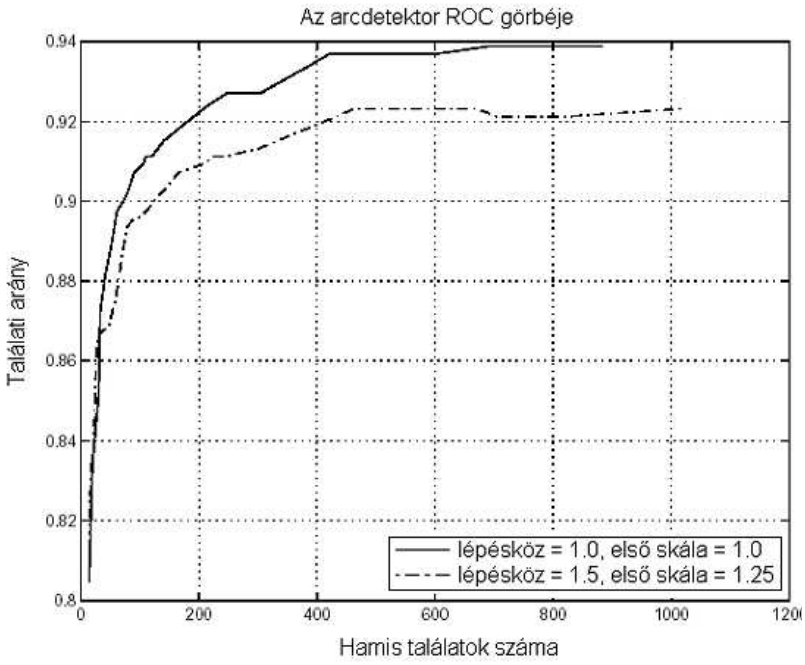
A 32 szint teljes tanítási ideje egy hét volt egy 466 MHz-es AlphaStation XP900-as gépen. A nehézkes tanítási folyamat során néhány javítást fedeztek fel a tanító algoritmushoz. Ezek a javítások a tanítási idő 100-szoros csökkenését eredményezték.

7.1.3. A végső detektor sebessége

A kaszkád detektor sebessége közvetlenül kapcsolatban áll az egy alablakon kiértékelt jellemzők számával. Ahogy azt a 2.3.1. fejezet is leírja a kiértékelt jellemzők száma a letapogatott képektől függ. Az MIT+CMU tesztadathalmazon átlag 8 jellemző kerül kiértékelésre a 4297 jellemzőből alablakonként. Ez lehetséges, mivel a képek nagy többsége már a kaszkád első vagy második szintjén visszautasításra kerül. A Rowley-Baluja-Kanade detektornál [3] körülbelül 15-ször, a Schneiderman-Kanade detektornál [4] pedig nagyjából 600-szor gyorsabb.

7.1.4. Kísérletek egy valós tesztadatbázison

Az alkotók az MIT+CMU frontális arcadatbázison tesztelték rendszerüket [3]. Ez az adatbázis 130 képet tartalmaz 507 frontális arccal. A 7.2. ábra mutatja az arcdetektor eredményeit néhány képen a tesztadatbázisból.



7.2. ábra. Az arcdetektor ROC görbéje az MIT+CMU tesztadatbázison. A detektort először 1.0 lépésközzel és 1.0 kezdeti skálával futtatták (így 75,081,800 alablakot tapogatott le a detektor), majd 1.5 lépésközzel és 1.25 kezdeti skálával futtatták (így 18,901,947 alablakot tapogatott le a detektor). Mindkét esetben 1.25-ös skálázási faktort használtak.

A ROC görbe létrehozásához az utolsó osztályozó perceptronjának küszöbét $+\infty$ és $-\infty$ között állították be. A küszöb $+\infty$-re állítása 0.0-ás detektálási arányt és 0.0-ás hamis detektálási arányt hoz. A küszöb $-\infty$-re állítása, habár növeli mind a detektálási arányt, mind a hamis detektálási arányt, de csak egy bizonyos pontig. Egyik arány sem lehet nagyobb mint a detektor kaszkádból a befejező szint eltávolításával kapott detektor kaszkád arányai. Tulajdonképpen egy $-\infty$ küszöb egyenlő ennek a szintnek az eltávolításával. A detektálási és a hamis detektálási arány további növeléséhez a kaszkád következő osztályozójának küszöbét kell csökkenteni. Így, hogy egy teljes ROC görbét állítsanak elő, eltávolítottak osztályozó szinteket. A hamis pozitív eredmények számát használják szemben a hamis pozitív aránnyal a ROC görbe x-tengelye mentén, hogy megkönnyítsék az összehasonlítást más rendszerekkel. A hamis pozitív arány kiszámításához egyszerűen osszuk le a hamis pozitív eredmények számát a letapogatott alablakok számával. A $\triangle = 1.0$ és kezdő skála $= 1.0$ esetben az alablakok száma 78,081,800. A $\triangle = 1.5$ és kezdő skála $= 1.25$ esetében pedig 18,901,947 a letapogatott alablakok száma. Mindkét esetben 1.25-ös skálázási faktort használtak.

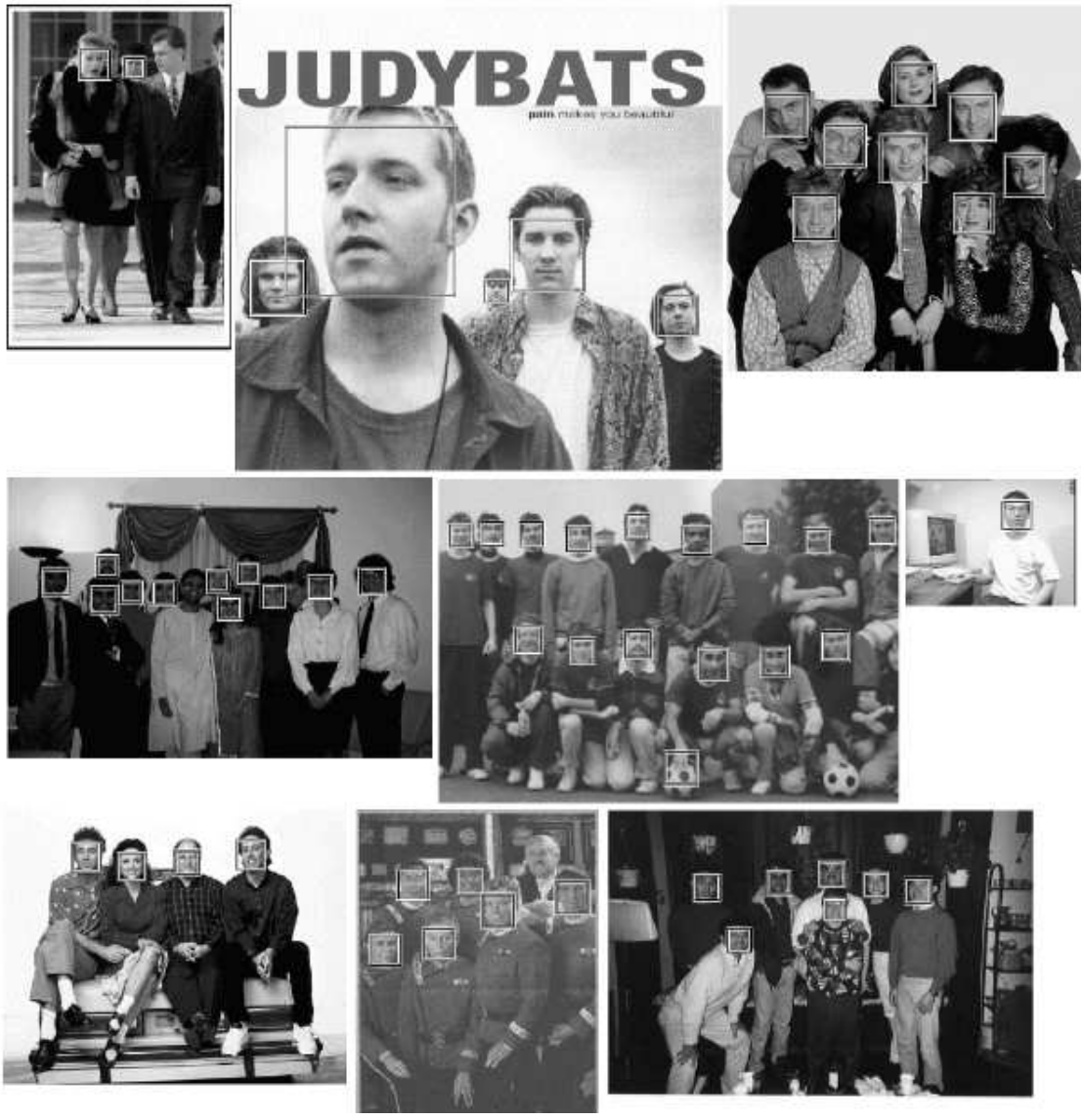
Sajnos az arcdetektálás előzőleg publikált eredményei csak egyszerű műveleti rendszereket foglal magába (pl. egyszerű pontokat a ROC görbén). Az eddigi publikációk hamis pozitív arányaihoz mérten mutatom meg az eredményeket azért, hogy könnyebben összehasonlíthassuk a Viola-Jones arcdetektort a többi arcdetektáló rendszerrel. A 7.1. táblázat mutatja a Viola-Jones arcdetektornak és más rendszereknek a detektálási arányait különböző hamis pozitív eredmények mellett. A Rowley-Baluja-Kanade detektor [3] különböző verzió különböző eredményeket mutatnak. A különböző eredmények tulajdonképpen nem pontok a ROC görbén, hanem azok közelítése. Két detektoruk ROC görbéjét publikálták, de ezek a ROC görbék nem a legjobb eredményeiket reprezentálják. Így a táblázatbeli detektálási arányok a Rowley-Baluja-Kanade detektornál tulajdonképpen detektoruk különböző verzióinak eredményei. A Roth-Yang-Ahuja detektor [5] eredményeit az MIT+CMU tesztadatbázisból 5 képet elhagyva kapták, a képek közül eltávolították a kézzel rajzoltakat. Így ők az eredményüket az MIT+CMU tesztadatbázis egy részhalmazán kapták, mely 125 képet tartalmazott 483 frontális arccal. Feltehetően a detektálási arányuk alacsonyabb lenne, ha a teljes tesztadatbázist használták volna. A zárójel az eredmény körül ezt az eltérést jelöli.

A Sung and Poggio arcdetektor az MIT+CMU adatbázis MIT részhalmazán tesztelték, ugyanis a CMU részhalmaz akkor még nem létezett. Az MIT részhalmaz 23 képet tartalmaz 149 arccal. 5 hamis pozitív eredmény mellett 79.9%-os detektálási arányt értek el. A Viola-Jones detektor eredménye 5 hamis pozitív mellett 77.8% az MIT részhalmazon.

A 7.3. ábra mutatja a Viola-Jones detektor kimenetét az MIT+CMU tesztadatbázis néhány képén.

Hamis detektálások / **Detektorok**	10	31	50	65	78	95	110	167	422
Viola-Jones	78.3%	85.2%	88.8%	89.8%	90.1%	90.8%	91.1%	91.8%	93.7%
Rowley-Baluja-Kanade	83.2%	86.0%	-	-	-	89.2%	-	90.1%	89.9%
Schneiderman-Kanade	-	-	-	94.4%	-	-	-	-	-
Roth-Yang-Ahuja	-	-	-	-	(94.8%)	-	-	-	-

7.1. táblázat. Detektálási arány különböző hamis pozitív értékek esetén az MIT+CMU tesztadatbázison, amely 130 képet tartalmaz 507 frontális arccal.



7.3. ábra. Az arcdetektor eredménye az MIT+CMU adatbázisban néhány képen demonstrálva.

7.2. Szemdetektorok teszteredményei

Szintek	Találatok		Hiányzó találatok		Hamis találatok		Gyenge osztályozók		Találati arány (%)	
	30x15	40x20	30x15	40x20	30x15	40x20	30x15	40x20	30x15	40x20
25	419		31		132		340		93,11	
24	422		28		144		333		93,77	
23	423	418	27	32	155	613	323	217	94	92,88
22	423	424	27	26	243	758	275	208	94	94,22
21	426	426	24	24	402	881	248	198	94,66	94,66
20	432	427	18	23	662	1044	228	181	96	94,88
19	433	429	17	21	1116	1582	205	166	96,22	95,33
18	436	429	14	21	1866	2457	185	150	96,88	95,33
17	442	436	8	14	2744	4085	169	135	98,22	96,88
16	442	437	8	13	4157	7202	152	123	98,22	97,11
15	443	438	7	12	7550	9537	132	108	98,44	97,33
14	443	444	7	6	11275	14106	116	97	98,44	98,66
13	444	446	6	4	18782	18585	101	85	98,66	99,11
12	445	441	5	9	24158	24666	89	73	98,88	98
11	444	441	6	9	33509	30909	76	60	98,66	98

7.2. táblázat. Szemkörnyék detektorok eredményei. Az egyes detektorok:
30x15. eyes2-30x15-3547-450-sym
40x20. eyes2-40x20-3547-450-sym

Szintek		Találatok						Hiányzó találatok						Találati arány (%)				
		1.	2.	3.	4.	5.		1.	2.	3.	4.	5.		1.	2.	3.	4.	5.
25		430	418	411	406	405		32	32	39	44	45		95	92,88	91,33	90,22	90
24		430	421	416	408	407		29	29	34	42	43		95	93,55	92,44	90,66	90,44
23		432	421	418	412	409		29	29	32	38	41		96	93,55	92,88	91,55	90,88
22		432	424	420	411	409		26	26	30	39	41		96	94,22	93,33	91,33	90,88
21		432	426	418	411	410		24	24	32	39	40		96	94,66	92,88	91,33	91,11
20		432	428	419	421	411		22	22	31	29	39		96	95,11	93,11	93,55	91,33
19		432	427	416	420	414		23	23	34	30	36		96	94,88	92,44	93,33	92
18		430	428	422	421	411		22	22	28	29	39		95	95,11	93,77	93,55	91,33
17		436	435	421	421	412		15	15	29	29	38		96,88	96,66	93,55	93,55	91,55
16		434	436	421	419	410		14	14	29	31	40		96,44	96,88	93,55	93,11	91,11
15		430	434	423	418	409		16	16	27	32	41		95	96,44	94	92,88	90,88
14		431	435	422	416	415		15	15	28	34	35		95,77	96,66	93,77	92,44	92,22
13		431	432	420	415	410		18	18	30	35	40		95,77	96	93,33	92,22	91,11
12		429	431	413	414	404		19	19	37	36	46		95,33	95,77	91,77	92	89,77
11		432	433	419	410	408		17	17	31	40	42		96	96,22	93,11	91,11	90,66

7.3. táblázat. Balszem detektorok teszteredményei. Az egyes detektorok:
1. left2-25x20-3545-450-sym,
2. left2-25x20-3545-450-nonsym,
3. left3-25x15-7097-450-sym,
4. left3-25x15-7097-450-nonsym,
5. left3-25x15-7097-855-nonsym.

Szintek		Hamis találatok						Gyenge osztályozók				
		1.	2.	3.	4.	5.		1.	2.	3.	4.	5.
25		1978	795	1062	771	478		679	477	953	609	742
24		2473	944	1192	855	503		637	449	897	576	688
23		2812	996	1391	994	600		600	427	829	540	636
22		3091	1182	1739	1142	643		561	402	768	504	583
21		3991	1496	2037	1382	731		518	378	707	470	536
20		4476	2005	2450	2177	898		476	349	652	434	490
19		5604	2499	2889	2716	1187		436	323	602	403	444
18		7147	3693	3893	3330	1585		398	293	551	372	403
17		9088	5872	5498	4310	2002		357	269	503	337	361
16		12631	8716	8234	6334	2744		319	243	450	307	328
15		16136	11020	11296	9296	3669		280	213	397	277	285
14		20055	15634	14922	13340	7180		243	186	348	248	253
13		25756	22814	18196	16659	11817		208	165	301	218	220
12		32059	28179	23022	21537	16660		174	141	260	184	190
11		42706	37104	29769	26390	29376		146	118	223	158	162

7.4. táblázat. Balszem detektorok teszteredményei - folytatás.

Szintek		Találatok				Hiányzó találatok				Találati arány (%)		
		1.	2.	3.		1.	2.	3.		1.	2.	3.
25		430	415	407		35	20	43		95,55	92,22	90,44
24		432	414	408		36	18	42		96	92	90,66
23		432	419	410		31	18	40		96	93,11	91,11
22		433	422	406		28	17	44		96,22	93,77	90,22
21		433	422	404		28	17	46		96,22	93,77	89,77
20		434	423	407		27	16	43		96,44	94	90,44
19		431	424	414		26	19	36		95,77	94,22	92
18		429	421	413		29	21	37		95,33	93,55	91,77
17		431	423	416		27	19	34		95,77	94	92,44
16		431	420	411		30	19	39		95,77	93,33	91,33
15		433	418	411		32	17	39		96,22	92,88	91,33
14		434	416	416		34	16	34		96,44	92,44	92,44
13		430	421	415		29	20	35		95,55	93,55	92,22
12		431	420	412		30	19	38		95,77	93,33	91,55
11		432	419	418		31	18	32		96	93,11	92,88

7.5. táblázat. Jobbszem detektorok teszteredményei. Az egyes detektorok:
1. right2-25x20-3563-450-nonsym,
2. right3-25x15-7041-450-nonsym,
3. right3-25x15-7041-855-nonsym.

Szintek		Hamis találatok				Gyenge osztályozók		
		1.	2.	3.		1.	2.	3.
25		2062	862	513		497	578	1045
24		2315	1054	577		470	541	966
23		2914	1342	698		443	508	895
22		4093	1602	892		410	473	817
21		5087	1795	1049		378	439	756
20		6108	2435	1274		351	404	692
19		7625	3062	1761		323	374	631
18		9810	3779	2944		294	338	566
17		12583	4641	4203		264	306	508
16		16603	6025	5527		238	276	454
15		21200	7533	8508		213	246	394
14		25908	12989	12743		191	214	342
13		32461	16999	15912		165	186	296
12		37521	22067	21917		144	160	259
11		42381	28047	25559		123	137	218

7.6. táblázat. Jobbszem detektorok teszteredményei - folytatás.

Szintek	Találatok		Hiányzó találatok		Hamis találatok		Gyenge osztályozók		Találati arány (%)	
	30x15	40x20	30x15	40x20	30x15	40x20	30x15	40x20	30x15	40x20
25	410	21			95,12		31		355	
24	412	19			95,59		34		340	
23	414	17	28	403	96	93,5	39	120	333	222
22	414	17	27	404	96	93,73	45	127	323	217
21	414	17	25	406	96	94,19	69	143	275	208
20	415	16	24	407	96,2	94,43	105	157	248	198
19	414	17	21	410	96	95,12	168	188	228	181
18	420	11	19	412	97,44	95,59	262	298	205	166
17	424	7	14	417	98,37	96,75	436	470	185	150
16	424	7	10	421	98,37	97,67	622	798	169	135
15	424	7	9	422	98,37	97,91	923	1299	152	123
14	428	3	7	424	99,3	98,37	1661	1762	132	108
13	431	0	3	428	100	99,3	2541	2788	116	97
12	431	0	2	429	100	99,53	4224	4009	101	85
11	429	2	3	428	99,53	99,3	5844	5096	89	73
10	429	2	3	428	99,53	99,3	9446	6622	76	60

7.7. táblázat. Szemkörnyék detektorok hierarchikus teszteredményei. Az egyes detektorok:
30x15. eyes2-30x15-3547-450-sym
40x20. eyes2-40x20-3547-450-sym

Szintek		Találatok						Hiányzó találatok						Találati arány (%)				
		1.	2.	3.	4.	5.		1.	2.	3.	4.	5.		1.	2.	3.	4.	5.
25		412	408	415	407	405		23	27	20	28	30		94,71	94,44	95,4	93,56	93,1
24		411	405	415	409	408		24	30	20	26	27		94,48	93,1	95,4	94,02	93,79
23		414	404	415	409	409		21	31	20	26	26		95,17	92,87	95,4	94,02	94,02
22		415	405	419	411	409		20	30	16	24	26		95,4	93,1	96,32	94,48	94,02
21		416	408	421	412	409		19	27	14	23	26		95,63	94,44	96,78	94,71	94,02
20		419	409	421	415	410		16	26	14	20	25		96,32	94,02	96,78	95,4	94,25
19		419	415	421	425	414		16	20	14	10	21		96,32	95,4	96,78	97,7	95,17
18		420	415	420	426	417		15	20	15	9	18		96,55	95,4	96,55	97,93	95,86
17		421	417	422	426	418		14	18	13	9	17		96,78	95,86	97,01	97,93	96,09
16		422	422	426	427	419		13	13	9	8	16		97,01	97,01	97,93	98,16	96,32
15		424	420	428	429	421		11	15	7	6	14		97,47	96,55	98,39	98,62	96,78
14		422	419	430	432	423		13	16	5	3	12		97,01	96,32	97,7	99,31	97,24
13		424	419	432	433	426		11	16	3	2	9		97,47	96,32	99,31	99,54	97,93
12		425	421	433	433	430		10	14	2	2	5		97,7	96,78	99,54	99,54	98,85
11		431	427	434	432	434		4	8	1	3	1		99,08	98,16	99,77	99,31	99,77
10		433	429	434	434	435		2	6	1	1	0		99,54	98,62	99,77	99,77	100

7.8. táblázat. Balszem detektorok hierarchikus teszteredményei. Az egyes detektorok:
1. left2-25x20-3545-450-sym,
2. left2-25x20-3545-450-nonsym,
3. left3-25x15-7097-450-sym,
4. left3-25x15-7097-450-nonsym,
5. left3-25x15-7097-855-nonsym.

Szintek		Hamis találatok						Gyenge osztályozók				
		1.	2.	3.	4.	5.		1.	2.	3.	4.	5.
25		64	39	12	7	1		722	504	1004	646	788
24		69	48	17	6	0		679	477	953	609	742
23		76	48	16	9	0		637	449	897	576	688
22		77	50	18	11	0		600	427	829	540	636
21		81	51	21	10	0		561	402	768	504	583
20		90	53	23	14	1		518	378	707	470	536
19		92	45	34	26	3		476	349	652	434	490
18		115	56	46	33	7		436	323	602	403	444
17		142	76	56	49	14		398	293	551	372	403
16		155	97	75	78	20		357	269	503	337	361
15		177	125	132	126	31		319	243	450	307	328
14		240	157	177	173	62		280	213	397	277	285
13		301	222	243	251	120		243	186	348	248	253
12		397	340	301	305	270		208	165	301	218	220
11		508	433	460	482	367		174	141	260	184	190
10		746	634	601	687	592		146	118	223	158	162

7.9. táblázat. Balszem detektorok hierarchikus teszteredményei - folytatás.

Szintek		Találatok				Hiányzó találatok				Találati arány (%)		
		1.	2.	3.		1.	2.	3.		1.	2.	3.
25		404	408	405		31	27	30		92,87	93,79	93,1
24		405	409	408		30	26	27		93,1	94,02	93,79
23		408	410	411		27	25	24		93,79	94,25	94,48
22		405	417	414		30	18	21		93,1	95,86	95,17
21		408	417	415		27	18	20		93,79	95,86	95,4
20		410	418	416		25	17	19		94,25	96,09	95,63
19		410	425	418		25	10	17		94,25	97,7	96,09
18		413	427	422		22	8	13		94,94	98,16	97,01
17		418	429	426		17	6	9		96,09	98,62	97,93
16		420	429	426		15	6	9		96,55	98,62	97,93
15		419	429	427		16	6	8		96,32	98,62	98,16
14		424	430	427		11	5	8		97,47	98,85	98,16
13		430	430	432		5	5	3		98,85	98,85	99,31
12		428	432	433		7	3	2		98,39	99,31	99,54
11		429	432	432		6	3	3		98,62	99,31	99,31
10		430	432	431		5	3	4		98,85	99,31	99,08

7.10. táblázat. Jobbszem detektorok hierarchikus teszteredményei. Az egyes detektorok:
1. right2-25x20-3563-450-nonsym,
2. right3-25x15-7041-450-nonsym,
3. right3-25x15-7041-855-nonsym.

Szintek		Hamis találatok				Gyenge osztályozók		
		1.	2.	3.		1.	2.	3.
25		72	8	2		531	618	1127
24		78	12	3		497	578	1045
23		90	19	4		470	541	966
22		100	21	12		443	508	895
21		119	31	15		410	473	817
20		131	33	15		378	439	756
19		149	63	22		351	404	692
18		177	86	35		323	374	631
17		227	101	70		294	338	566
16		263	120	116		264	306	508
15		298	139	171		238	276	454
14		365	169	246		213	246	394
13		526	300	378		191	214	342
12		701	413	470		165	186	296
11		883	561	578		144	160	259
10		989	707	734		123	137	218

7.11. táblázat. Jobbszem detektorok hierarchikus teszteredményei - folytatás.

7.3. Hough transzformáció

A Hough transzformáció a képelemzés gyakori eszköze globális mintafelismerésre egy képtéren belül [19, 20, 21, 22, 23]. Ezt lokális mintafelismeréssel valósítja meg a transzformált paramétertérben.

A technika alapötlete, hogy keressünk olyan görbéket, melyek egy megfelelő paramétertérben az egyenes vonalakhoz, a polinomokhoz, a körökhöz, stb. hasonlóan paraméterezhetők. A transzformáció magasabb dimenziókban is alkalmazható, mégis a fő használati terület két dimenzióban egyenesek keresése, adott sugarú körök középpontjának keresése, $y = ax^2 + bx + c$ alakú parabolák keresése, ahol c konstans, stb.

A transzformáció éldetektált képeken működik.

7.3.1. Egyenes keresés Hough transzformációval

A legegyszerűbb esetben egyeneseket kereshetünk a transzformációval. Éldetektálás után egy bináris képet kapunk, ahol azok a pontok szerepelnek majd, melyek nagy eséllyel valamilyen élen fekszenek.

Egy egyenest felírhatunk a következő alakban:

$$y = mx + b,$$

ahol m a meredekség, b pedig az eltolás mértéke, vagyis ahol az egyenes metszi az y tengelyt. Egy adott (x, y) ponton, ilyen alakú egyenesek mehetnek át. Az egyenes egyenlete felírható a következő alakban is:

$$b' = y - mx'.$$

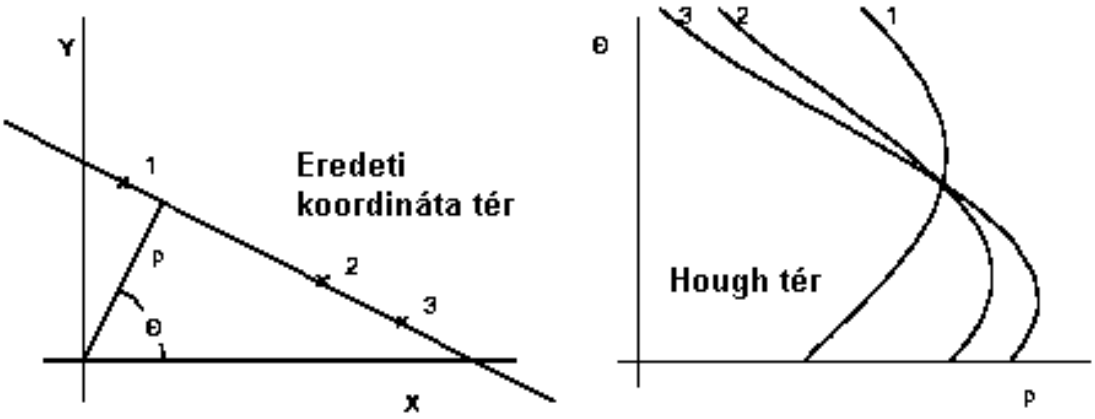
Ekkor m' változtatásával, megkapjuk a megfelelő b' paramétereket. Ezt minden éldetektált pontra elvégezzük az éldetektált képen. Egy (m, b) akkumulátor mátrixban pedig mindig növeljük a megfelelő (m',b') pontot. Végül az (m, b) akkumulátor mátrix maximuma fogja megadni a keresett egyenes egyenletének paramétereit.

Ez a módszer azonban nem elég megbízható, mivel m és b értéke a végtelenbe tart. Ehelyett az egyenesek reprezentációjára a következő alakot alkalmazzuk:

$$\rho = x \cos \Theta + y \sin \Theta,$$

ahol ρ az egyenes origótól vett merőleges távolsága, Θ pedig a normálvektor és az origó által bezárt szög (7.4. ábra). Minden éldetektálás után kapott (x, y) pontra elvégzünk egy az előzőhöz hasonló iterációt. Θ-t iteráljuk 0 és 360 fok között, ρ-t pedig kiszámítjuk hozzá. Így a (ρ, Θ) paramétertérben most görbéket fogunk kapni. Most egy (ρ, Θ) alakú akkumulátor mátrixunk lesz, mely elemeinek értékét növeljük, ha (x, y) ponton átmehet ilyen paraméterű egyenes. Ha az eredeti képen van egy $\rho' = x \cos \Theta' + y \sin \Theta'$ alakú egyenes szakasz, akkor a (ρ, Θ) paramétertér (ρ', Θ') pontjában több görbe is metszi egymást.

A (ρ, Θ) paramétertér felosztásával, kvantálásával finoman kell bánni. Ha a paramétertér felosztása túl finom, azaz a cellák túl szűkek, két-két szinuszos görbe metszete különböző cellákba kerülhet. Ha viszont a felosztás nem elég finom, párhuzamos egymáshoz közeli egyenesek egy cellába kerülhetnek a paramétertérben.



7.4. ábra. **Egyenes transzformálása Hough térbe**

A (ρ, Θ) akkumulátor mátrix maximumai azonosítják az egyenes szakaszokat az eredeti képen. Ideális esetben a Hough transzformáció egyetlen maximumot keres. Abban az esetben, ha a kép több különböző méretű mintát tartalmaz, és itt nem csak egyenesekre kell gondolni, megkereshetjük azokat a maximumokat, melyek leginkább mintát azonosítanak, majd megismételhetjük a folyamatot.

Vonal keresésre láthatunk példát a 7.5. és a 7.6. ábrákon.

7.5. ábra. **Vonal keresés Hough transzformációval**

7.6. ábra. **Vonal keresés Hough transzformációval.** Eredeti kép - éldetektált kép - Hough tér - legjobb egyenes.

7.3.2. Kör keresés Hough transzformációval

A kört a következő alakban írhatjuk fel:

$$(x-a)^2 + (y-b)^2 = r^2,$$

ahol r a kör sugara, (a, b) pedig a középpontja. Mivel itt 3 paraméter van, a Hough transzformált kép egy 3 dimenziós kép lesz. Ezért a kör keresés nagyobb számítás igényű, mint a vonal keresés. Mint az egyenes esetében, rögzítünk egy paramétert, és a többit változtatjuk amíg lehet. Utána a rögzített paramétert növeljük/csökkentjük, és újra a többi paramétert változtatjuk. Ha adott sugarú kört keresünk, akkor eggyel kevesebb paraméterünk lesz, ami gyorsítja a számítást.

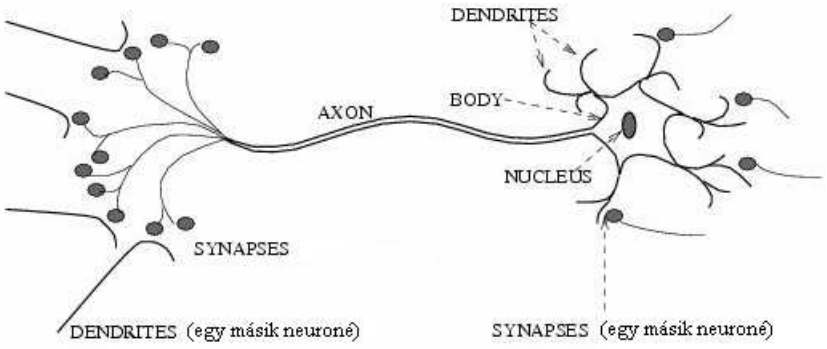
7.4. Neurális hálózatok

A fejezet rövid elméleti áttekintést nyújt a neurális hálózatok és a mesterséges neurális hálózatok elméletéről. A megfelelő irodalomban olvashatunk a fogalmak mélyebb magyarázatáról [30], a mesterséges neuronhálók matematikai hátteréről [31], egy pszichológiai és fiziológiai megközelítésről [32] és a gyakorlati szempontokról [33].

Az emberi agy egy bonyolult rendszer, mely nagyon összetett feladatok megoldására képes. Habár az agy működésének néhány alapvető műveletét már megfejtették, még mindig távol vagyunk a teljes működés megértésétől.

A mesterséges neuronhálók működésének megértéséhez szükséges az agy belső rendszerének alapvető ismerete. Az agy a központi idegrendszer része és egy nagy neurális hálózatból áll. A neuronháló valójában igen bonyolult, ezért csak azokra a részekre térek ki, melyek elmaradhatatlanok a működés megértéséhez.

A neuronháló kapcsolódó neuronokból áll. A neuron közepe egy sejttest, amit nucleus-nak nevezünk. A nucleus dendritek és axonok segítségével kapcsolódik egy másik nucleus-hoz. A neuron axonja és egy másik neuron dendritje közötti rést szinapszisnak nevezünk. A neuron elektromos ingereket bocsát ki a szinapszis kapcsolatain keresztül, amit egy másik neuron dendritjei fogadnak. A 7.7. ábra mutatja egy neuron felépítését.



7.7. ábra. Egy neuron képe.

Ha egy neuron elegendő elektromos ingerületet fogad a dendritjein keresztül, akkor aktiválódik és tüzelni fog az axonon át, az elektromos ingereit pedig egy másik

neuron fogadja majd. Az információ ilyen formában terjed a neuronhálóban. A neuron élettartama folyamán a szinapszis kapcsolatok mindvégig változnak és a tüzelést kiváltó beérkező ingerületek összege, a küszöb is változik. A neuronhálót ez a tulajdonsága teszi képessé a tanulásra. Az emberi agy körülbelül 10^{11} neuronból áll, melyek szorosan kapcsolódnak, a kapcsolatok száma körülbelül 10^{15} [33]. Ezek a neuronok párhuzamosan aktiválódnak külső és belső hatásokra. Az agy az idegrendszer többi részével is kapcsolatban áll, információkat fogad az öt érzékszerven keresztül és vezérli az izomzat működését.

7.4.1. Mesterséges neuronhálók

Jelenleg nem lehetséges egy mesterséges emberi agy elkészítése, de egy egyszerűsített mesterséges neuront és egy mesterséges neuronhálót tudunk készíteni. Egy mesterséges neurális hálózatot többféleképpen készíthetünk és az agy működését is többféleképpen imitálhatjuk.

A mesterséges neuronhálók jól teljesítenek mintafelismerésben és egyszerű szabályok készítésében bonyolult problémákhoz. A tanulási képességeik szintén kitűnőek, ezért a mesterséges intelligencia területén gyakran alkalmazzák őket.

A mesterséges neuronhálók képesek egy tanítóhalmazt jól általánosítani. Például ha egy mesterséges neuronhálónak megadjuk állatok egy halmazát és a tényt, hogy melyik emlős, melyik nem, akkor a háló képes megmondani egy a tanítóhalmazon kívüli állatról, hogy emlős vagy sem. Ez egy nagyon kívánatos jellemzője a mesterséges neuronhálóknak, mivel nem szükséges tudnunk, hogy mik az emlősök jellemzői, a neuronháló magától kitalálja azokat.

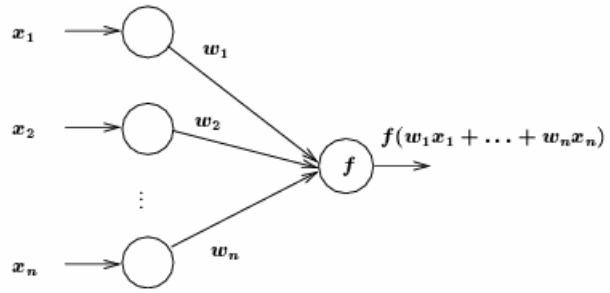
A mesterséges neuron

Egy egyszerű mesterséges neuront több módon is implementálhatunk. Az általános matematikai definíciója a következő (7.1):

$$y(x) = g\Big(\sum_{i=0}^{n} w_i x_i\Big), \tag{7.1}$$

ahol x egy neuron n bemenő dendrittel $(x_0 \dots x_n)$ és egy kimenő axonja van: $y(x)$, $(w_0 \dots w_n)$ pedig a bemenetek súlyai.

g az aktivációs függvény, ami meghatározza, milyen erős legyen a kimenet a bemenetek összege alapján. Ha a mesterséges neuron egy igazi neuront imitálna, akkor az aktivációs függvény g egy egyszerű küszöbölés kellene legyen 0 vagy 1 visszatérési értékkel. Mégis a mesterséges neuronokat rendszerint nem így implementálják. Különböző okokból jobb megoldást jelent egy egyenletes (lehetőleg differenciálható) aktivációs függvény használata. A aktivációs függvény kimeneti értéke vagy 0 és 1 között van, vagy -1 és 1 között, attól függően, hogy melyik aktivációs függvényt használjuk. Ez nem teljesen igaz, amióta például az identitás függvényt is használják aktivációs függvényként, aminek nincsenek ilyen korlátai, de a többi aktivációs függvényre teljesül ez a megszorítás. A bemenetekre és a súlyokra nincsenek megszorítások, értékük $-\infty$ és ∞ között lehet, mégis gyakran igen kis értéket vesznek fel a nulla közelében. A 7.8. ábra mutatja, hogyan néz ki egy mesterséges neuron.



7.8. ábra. Egy mesterséges neuron képe.

Egy igazi neuronban a súlyok megfeleltethetők a beérkező ingerületek számának, erősségének és a szinapszis kapcsolatok zártságának.

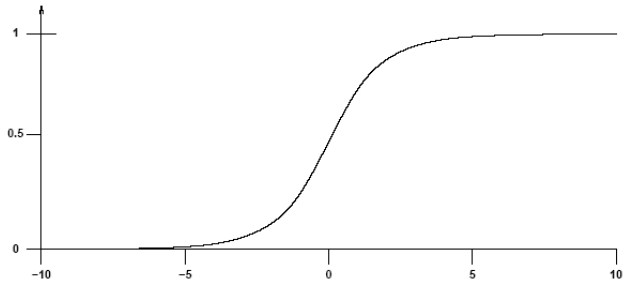
Ahogy azt már korábban említettem, több lehetséges aktivációs függvényt használhatunk. Néhány gyakran használt ezek közül a lépcső vagy küszöb (7.2), a szigmoid (7.3) és a tangens hiperbolikus függvény (7.4).

$$g(x) = \begin{cases} 1 & \text{ha } x + t > 0 \\ 0 & \text{ha } x + t \leq 0 \end{cases} \tag{7.2}$$

$$g(x) = \frac{1}{1 + e^{-2s(x+t)}} \tag{7.3}$$

$$g(x) = tanh(s(x+t)) = \frac{sinh(s(x+t))}{cosh(s(x+t))} = \frac{e^{s(x+t)} - e^{-s(x+t)}}{e^{s(x+t)} + e^{-s(x+t)}} = \frac{e^{2(s(x+t))} - 1}{e^{2(s(x+t))} + 1} \tag{7.4}$$

Ahol t az eltolás mértéke, s pedig a lépésköz paraméter. A szigmoid és a tangens hiperbolikus függvények egyenletesen differenciálhatók, függvény képük nagyon hasonló, az egyetlen fontos különbség közöttük, hogy a tangens hiperbolikus függvény -1 és 1 közötti értékeket vesz fel, a szigmoid függvény pedig 0 és 1 közötti értékeket. A szigmoid függvény képe látható a 7.9. ábrán.



7.9. ábra. A szigmoid függvény képe, $s = 0.5$ és $t = 0$.

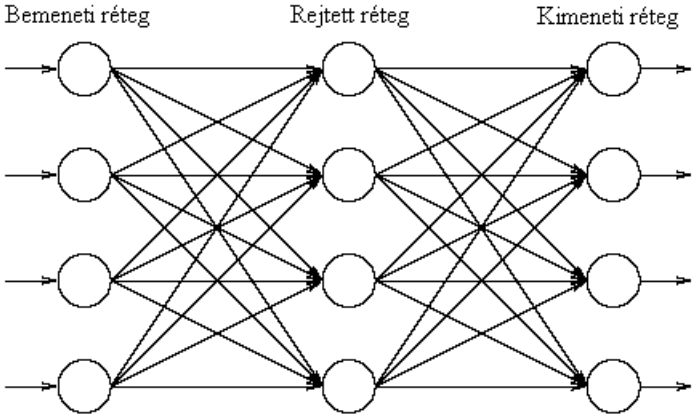
Egy mesterséges neuronban a t paraméter megfelel a neuron aktiváláshoz szükséges ingerek összegének. Ez a paraméter és a súlyok a tanulás folyamán kapnak értéket.

A mesterséges neuronháló

A leggyakrabban használt neurális hálózatok a többrétegű előrecsatolt mesterséges neuronhálók. Egy ilyen hálóban a neuronok rétegekbe rendeződnek, a háló egy bemeneti réteggel kezdődik és egy kimeneti réteggel fejeződik be. Eközött a két réteg között rejtett rétegek vannak. A hálóban csak előrecsatolások vannak az egyik rétegből a következő rétegbe. Sok másféle mesterséges neuronháló is létezik, de a következőkben a többrétegű előrecsatolt neurális hálózatokról lesz szó.

Ezeknek a hálóknak két fontos fázisa van, egy tanulási majd egy felhasználási fázisa. A tanulás során egy tanító halmazt dolgoz fel a neuronháló, melyben adott bemenet és kimenet párjaink vannak. A hálótól elvárjuk, hogy a tanítóhalmaz egy adott bemenetéhez a megfelelő kimenetet adja vissza. A felhasználási fázisban a kapott bemenet alapján valamilyen kimenetet ad a neuronháló.

Egy előrecsatolt neuronháló futtatása a következőképpen néz ki: a bemeneti réteg kap egy bemenetet, ami végigterjed a rétegeken (a 7.1 egyenlet szerint), amíg eléri a kimeneti réteget. Egy előrecsatolt neuronhálón könnyedén végigterjeszthetünk egy bemenetet és kiszámíthatjuk a hozzátartozó kimenetet. Egy hálóban, ahol a kapcsolatok minden irányban engedélyezettek (mint az emberi agyban), sokkal nehezebb a kimenet kiszámítása a lehetséges körök miatt. A körök felhasználására több lehetőség is van, [30] megmutatja, hogyan használhatók fel az időfüggő problémákban, míg az előrecsatolt hálók rendszerint jobb megoldást adnak az időfüggetlen problémákban.



7.10. ábra. Egy teljesen összekapcsolt többrétegű előrecsatolt neuronháló egy rejtett réteggel.

A 7.10. ábra egy többrétegű előrecsatolt neuronhálót mutat, ahol minden réteg minden neuronja kapcsolódik a következő réteg minden neuronjához. Az ilyen hálókat teljesen összekapcsoltnak nevezzük, és habár nem szükséges, hogy egy neuronháló teljesen összekapcsolt legyen, mégis gyakran használunk ilyen hálókat.

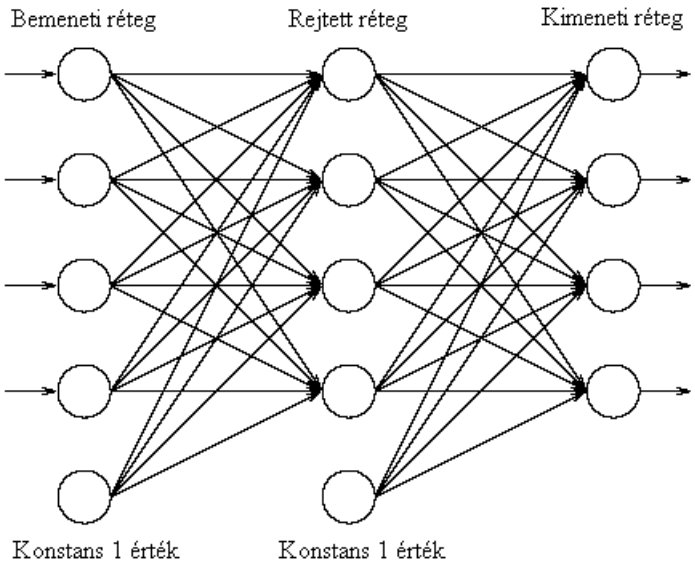
A tanítás során kétféle paramétert változtatunk, a súlyokat és a t értéket az aktivációs függvényben. Könnyebb lenne a tanítás, ha csak a súlyokat kellene beállítani. Ezért kezdtek el használni egy „konstans neuront”. Minden rétegben van egy neuron, amely a következő réteg összes neuronjával kapcsolódik, míg az előző rétegből eggyel

sem, és a kibocsátott értéke mindig 1. A hozzá kapcsolódó súlyt közvetlenül hozzáadjuk a többi neuron súlyokkal kombinált összegéhez (7.1. egyenlet), mintahogy t értéket az aktivációs függvényekben. A neuron egy módosított egyenlete tehát a következő lesz (7.5):

$$y(x) = g\left(\sum_{i=0}^{n+1} w_i x_i\right), \text{ ahol } x_{n+1} = 1 \text{ és } w_{n+1} = t. \tag{7.5}$$

A konstans érték használatával lehetővé válik, hogy elhagyjuk t értéket az aktivációs függvényből, és csak a súlyokat kell a tanítás folyamán beállítanunk. A szigmoid függvény egy módosított változatát mutatja a 7.6. egyenlet.

$$g(x) = \frac{1}{1 + e^{-2sx}} \tag{7.6}$$



7.11. ábra. Egy teljesen összekapcsolt többrétegű előrecsatolt neuronháló egy rejtett réteggel és konstans értékekkel.

Neuronhálók futásideje

Egy mesterséges neuronháló futásakor a 7.5. egyenletet számítjuk ki a háló minden neuronjára, kivéve a bemeneti és a konstans neuronokat. Ez azt jelenti, hogy egy szorzást és egy összeadást kell elvégeznünk minden kapcsolatban (az összeadás már a konstans értékekre is fenn áll), mielőtt kiszámítanánk az aktivációs függvény értékét minden neuronra, amely nem bemeneti neuron és nem a konstans érték. Ezek szerint a futási időt a következő egyenlet adja meg:

$$T = cA + (n - n_i)G \tag{7.7}$$

Ahol c a kapcsolatok száma, n az összes neuron száma, n_i a bemeneti és a konstans neuronok száma, A a szorzás költsége a neuron érték és a megfelelő súly között, és

még a szummához adást is ide számítjuk, G az aktivációs függvény költsége, T pedig a teljes költség.

Ha a neuronháló teljesen összekapcsolt, l a rétegek száma, n_l az egyes rétegekhez tartozó neuronok száma a konstans neuront leszámítva, akkor az előző egyenletet a következőképpen is írhatjuk:

$$T = (l-1)(n_l^2 + n_l)A + (l-1)n_l G \tag{7.8}$$

Az egyenlet megmutatja, hogy egy teljesen összekapcsolt neuronhálóban a futás idő A értékétől függ, tehát A értékét kell optimalizánunk.

7.4.2. Mesterséges neuronhálók tanítása

A tanításhoz bemeneti és kimeneti tanítópárok egy halmazára van szükség, és a tanítás folyamán azt szeretnénk elérni, hogy a súlyokat sikerüljön úgy beállítani, hogy a tanítóhalmaz egy bemenetéhez a megfelelő kimenetet adja a háló. Másrészt azt is fontos a tanítás folyamán, hogy ne csak a tanítóhalmaz bemeneteire adjon pontos eredményt a neuronhálónk, míg a tanítóhalmazon kívüliekre rosszat. Ha mégis ez történik, a neuronháló túltanulásáról beszélhetünk.

A tanítási folyamatot tekinthetjük egy optimalizációs problémának, ahol a teljes tanítóhalmaz feletti átlagos négyzetes hiba minimalizálására törekszünk. A leggyakrabban használt algoritmus a backpropagation (vagy hibavisszaterjesztő) algoritmus, de ennek az algoritmusnak van néhány korlátja az egyes iterációs lépések alatti súlybeállításra vonatkozóan. Erre a problémára ad megoldást néhány haladóbb algoritmus, mint például az RPROP [34] és a quickprop [35] algoritmus.

Backpropagation algoritmus

A hibavisszaterjesztő algoritmus a nevének megfelelően dolgozik, miután végigterjedt a neuronhálón a bemenet, kiszámítjuk a hibát, majd az visszaterjed a hálóba, a súlyok pedig ezalapján olyan új értéket vesznek fel, ami csökkenti a hibát. Az algoritmust most teljesen összekapcsolt neuronhálókra fogom bemutatni, de az elmélet ugyanez ritkán kapcsolódó hálók esetében is.

Habár a négyzetes hibát az összes tanító adatra szeretnénk minimalizálni, a leghatékonyabb mód ennek elérésére a backpropagation algoritmus használata mellett, ha az adatokat szekvenciálisan tanítjuk meg a hálónak, egyszerre csak egyet, ahelyett, hogy kombinálnánk a tanító adatokat. Ez azt jelenti, hogy a tanító adatok sorrendje számítani fog, ami nem feltétlenül előnyös, ugyanakkor elkerülhető, hogy az algoritmus lokális minimumokban elakadjon.

A backpropagation algoritmus először végigterjeszti a bemeneti adatot a neuronhálón. Ezután a kimenet k. neuronjára kiszámítható a hiba a következőképp:

$$e_k = d_k - y_k, \tag{7.9}$$

ahol y_k a k. neuron kapott kimenete, d_k pedig az elvárt kimenete. A hiba érték alapján kiszámítható a δ_k érték, melyet a súlyok további módosításához fogunk használni. A δ_k értéket a következő egyenlet alapján számítjuk ki:

$$\delta_k = e_k g'(y_k), \tag{7.10}$$

ahol g' a derivált aktivációs függvény. Ezért szükséges, hogy az aktivációs függvény differenciálható legyen.

Ha δ_k értéket kiszámítottuk, abból meghatározható δ_j a megelőző rétegekhez is a következő egyenlet alapján:

$$\delta_j = \eta \; g'(y_j) \sum_{k=0}^{K} \delta_k w_{jk}, \tag{7.11}$$

ahol K a neuronok száma ebben a rétegben és η a tanulási ráta, vagy más néven lépésköz, ami nagyban befolyásolja a tanulás hatékonyságát, ha túl nagyra választjuk, az algoritmus nem biztos, hogy eljut a minimumba, ha túl kicsire, akkor nagyon lassú lesz a tanulás folyamata.

A δ érték alapján kiszámíthatók a Δw értékek, amikkel a súlyokat módosítani fogjuk:

$$\Delta w_{jk} = \delta_j \; y_k. \tag{7.12}$$

A Δw_{jk} alapján az új súlyok, az új w_{jk}-k a következők lesznek: $w_{jk} = w_{jk} + \Delta w_{jk}$. Az algoritmus ezután veszi a következő bemenetet, és a súlyokat ismét beállítja az elvárt kimenetnek megfelelően. Ez a folyamat addig folytatódik, míg el nem érünk egy megállási kritériumot. A megállási kritérium rendszerint az átlagos négyzetes hiba mérésén alapul, ha ez a hiba elér egy bizonyos határt, akkor a tanuló folyamat leáll.