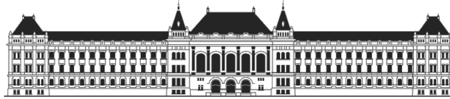


Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Irányítástechnika és Informatika Tanszék

Deep Learning a Vizuális Informatikában

Jegyzet

Szemenyei Márton
Varnyú Dóra

2023. december 5.

Tartalomjegyzék

I. Deep Learning Alapjai 6

1. Bevezetés a Vizuális Informatikába 7

1.1. Mi az a vizuális informatika? 7

1.1.1. Alapvető feladatok és nehézségek 8

1.2. Képfeldolgozás 9

1.3. Képi zajok, zajtípusok 10

1.3.1. Konvolúciós szűrések 10

1.3.2. Élesítés, élkeresés 13

1.4. Template matching 16

1.5. Hatékony konvolúció 18

1.5.1. Fourier-transzformáció 19

2. Neurális hálózatok 22

2.1. Bevezetés 22

2.2. Tanuló algoritmusok felépítése 22

2.2.1. Tanulás típusai 23

2.2.2. Nehézségek 23

2.3. Képosztályozás 24

2.3.1. Legközelebbi szomszéd 25

2.3.2. Lineáris osztályozás 25

2.4. A Perceptron tanítása 26

2.4.1. Hibafüggvények 27

2.4.2. Regularizáció 28

2.5. Optimalizáció 28

2.5.1. Gradiens alapú módszerek 29

2.5.2. Adaptív momentum (Adam) módszer 30

2.5.3. Másodrendű módszerek 30

2.5.4. Backpropagation 31

3. Konvolúciós Neurális Hálók 34

3.1. Bevezetés . . . 34

3.2. Konvolúciós neurális hálók . . . 34

3.2.1. Konvolúciós réteg . . . 34

3.2.2. Pooling . . . 35

3.2.3. Aktivációk . . . 37

3.3. Architektúrák . . . 38

3.3.1. AlexNet . . . 39

3.3.2. VGG . . . 39

3.3.3. Inception . . . 39

3.3.4. ResNet . . . 41

3.3.5. DenseNet . . . 42

3.4. CapsNet . . . 42

3.5. Vizualizáció . . . 46

3.5.1. Guided backpropagation . . . 46

3.5.2. Ellenséges példák . . . 47

3.5.3. DeepDream . . . 48

4. Deep Learning a gyakorlatban 50

4.1. Bevezetés . . . 50

4.2. Konvergencia problémák . . . 50

4.2.1. Inicializáció . . . 51

4.2.2. Adatnormalizálás . . . 51

4.3. Validáció és regularizáció . . . 52

4.3.1. Dropout . . . 53

4.3.2. Batch Normalization . . . 53

4.3.3. Adat Augmentáció . . . 54

4.4. Hiperparaméter optimalizáció . . . 55

4.4.1. Bayesi optimalizáció . . . 55

4.4.2. Tanulási ráta . . . 56

4.5. Adatbázisok előállítása . . . 57

4.5.1. Transfer learning . . . 57

4.5.2. Meta learning . . . 58

4.6. Installáció . . . 58

4.6.1. Pruning . . . 59

4.6.2. Weight sharing . . . 60

4.6.3. Ensemble . . . 61

4.6.4. Szeparálható konvolúció . . . 61

II. Magasszintű Látási Feladatok 63

5. RNN és Transzformer 64
5.1. Magasszintű látási feladatok 64
5.2. Visszacsatolt neurális hálók 66
5.2.1. LSTM 67
5.2.2. Alkalmazási példák 68
5.3. Transzformerek 69
5.3.1. Figyelem mechanizmus 69
5.3.2. A transzformer architektúra 72
5.3.3. A Vision Transformer 72

6. Detektálás és követés 76
6.1. Detektálás 76
6.1.1. Lokalizáció 76
6.1.2. Régió-CNN 76
6.1.3. YOLO 78
6.1.4. Anchor Free 80
6.2. Detektálás Transformerek segítségével 80
6.3. Követés 81
6.3.1. DeepSort 82
6.3.2. Követés Transzformerekkel 82
6.4. Fontos mérőszámok 83

7. Szegmentálás és videóanalízis 86
7.1. Szemantikus szegmentálás 86
7.1.1. Teljesen konvolúciós architektúra 86
7.1.2. Felskálázás módszerei 88
7.1.3. CRF 90
7.1.4. Költségfüggvények 92
7.2. Egyéb szegmentációs módszerek 93
7.2.1. Mask-R-CNN 93
7.3. Videók feldolgozása 94
7.3.1. Multi-frame hálók 94
7.3.2. Többfelbontású hálók 94
7.3.3. 3D konvolúció 95
7.3.4. Multimodális hálók 96
7.4. Videó osztályozás 98

8. 3D Feldolgozás **100**
8.1. Bevezetés . . . 100
8.2. 3D Struktúra reprezentációja . . . 100
8.2.1. Kd-fa reprezentáció . . . 101
8.3. 3D információ előállítása . . . 102
8.3.1. Többnézetű rekonstrukció . . . 102
8.3.2. Egynézetű mélységbecslés . . . 103
8.4. 3D adatok feldolgozása . . . 104
8.4.1. Voxel hálók . . . 105
8.4.2. Projekción alapuló hálók . . . 106
8.4.3. Pontfelhő hálók . . . 106
8.4.4. Kd-fa hálók . . . 106
8.4.5. MeshNet . . . 107
8.5. Spatial Transformers . . . 108

III. Felügyelet Nélküli Tanulás **110**

9. Generatív Hálók **111**
9.1. Bevezetés . . . 111
9.2. Stílus átültetés . . . 111
9.2.1. Jellemző rekonstrukció . . . 112
9.2.2. Textúra szintézis . . . 112
9.2.3. Neural Style Transfer . . . 113
9.3. Generatív modellek . . . 114
9.3.1. PixelRNN . . . 115
9.3.2. Variációs Autoencoder . . . 115
9.3.3. GAN . . . 117

10. Megerősítéses Tanulás **125**
10.1. Q Learning . . . 126
10.1.1. Markov döntési folyamat . . . 126
10.1.2. Érték és Q függvény . . . 127
10.1.3. Bellman szabály . . . 127
10.1.4. DQN . . . 128
10.2. Policy Gradiens (REINFORCE) . . . 129
10.2.1. Gradiens meghatározása . . . 129
10.2.2. Zajcsökkentés és baseline . . . 131
10.2.3. Actor-critic hálók . . . 131
10.2.4. Ritka jutalom és kíváncsiság . . . 132
10.3. Kemény figyelem . . . 134

11. Ön-felügyelt Tanulás **136**

11.1. Pre-text feladatok 136

11.1.1. Kép alapú módszerek 136

11.1.2. Videó alapú módszerek 138

11.2. Generatív módszerek 140

11.2.1. Autoencoderek 140

11.2.2. Bidirectional GAN 141

11.3. Kontrasztív tanulás 141

11.3.1. Sziámi hálók 142

11.3.2. SimCLR 143

11.3.3. MoCo 144

11.3.4. DINO 144

11.3.5. CPC 146

11.4. Few-shot tanulás 146

IV. Alkalmazások **148**

12. Neurális renderelés **149**

12.1. Hagyományos renderelés 149

12.2. Neurális renderelés 150

12.2.1. Új nézetek szintézise 151

12.2.2. Hatékonyabb reprezentáció 152

12.2.3. Deformálható színterek 152

12.2.4. Kontrollálható paraméterek 153

12.2.5. Dinamikus adatokra 154

12.3. Alkalmazások 154

12.3.1. Újravilágítás 155

12.3.2. Arc újraalkotás 156

12.3.3. Test újraalkotás 156

13. Orvosi alkalmazások **158**

13.1. Zajcsökkentés 158

13.2. Detekció és osztályozás 159

13.3. Szegmentáció 159

13.4. Képregisztráció és -fúzió 160

13.5. Kihívások 162

I. rész

Deep Learning Alapjai

1. fejezet

Bevezetés a Vizuális Informatikába

1.1. Mi az a vizuális informatika?

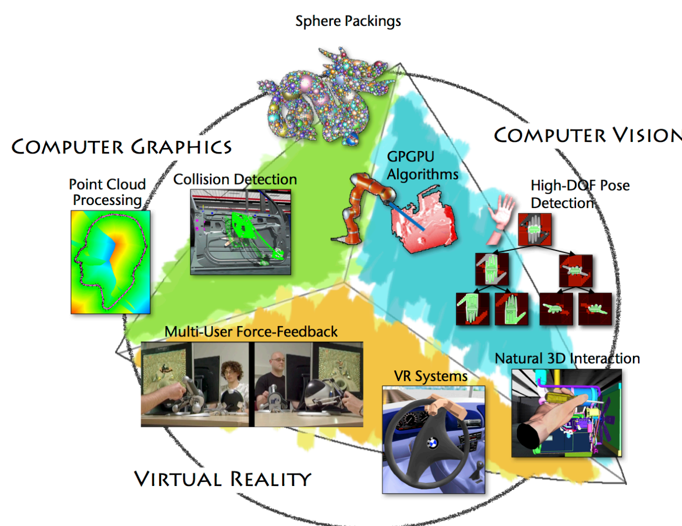
A vizuális informatika (visual computing) alkalmazásai az élet egyre több területén váltak elterjedtté. Ez a rendelkezésre álló számítási teljesítmény és az eszközök rohamos gyarapodásának, valamint a rendszerekben használt algoritmusok jelentős fejlődésének is köszönhető. A tudományterületnek számos felhasználási területe létezik: a robotikától és az automatizálástól kezdve a virtuális- és kiterjesztettvalóság-rendszereken keresztül egészen a szórakoztatóiparig.

A vizuális informatika két legalapvetőbb részterülete a számítógépes látás és grafika ágazatai. A két részterület egymással ellentétes feladatot lát el: míg a grafika során a világ(részlet) tartalmát, kinézetét és geometriáját leíró magasszintű adatokból egy szintetikus képet kívánunk előállítani, addig a látás során ezt pont fordítva kívánjuk elvégezni: egy természetes képből szeretnénk egy magasszintű leírást előállítani.

Az elvégzendő feladatok egyik legnagyobb nehézsége, hogy akár egyetlen kép is milliós nagyságrendű adatból (képpontból) áll, amelyek ráadásul ennél is számos nagyságrenddel több konfigurációban alkothatnak képeket. Azonban a valószerű képeket eredményező konfigurációkból is lényegesen kevesebb létezik, mint ahány lehetséges pixelkonfigurációt előállíthatunk, így nem lehetséges véletlenszerű pixelértékekkel képeket generálni. Ilyen adatmennyiség feldolgozásához vagy előállításához hatékony algoritmusokra és nagy teljesítményű eszközökre van szükség.

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

A számítógépes látáson belül gyakorta meg szoktuk különböztetni azokat a megoldásokat, amelyek a gépi tanulás (angolul: machine learning) algoritmusait használják a működésük során, és ezt a területet tanuló látásként (angolul: learning vision) is hívjuk. Ezeken belül külön figyelmet érdemelnek azok a megoldások, amelyek az elmúlt néhány évben rendkívül népszerűvé vált mélytanulás (angolul: deep learning) megoldásait alkalmazzák. A terület többi módszerét – sokszínűségük ellenére – hagyományos látásnak nevezzük.

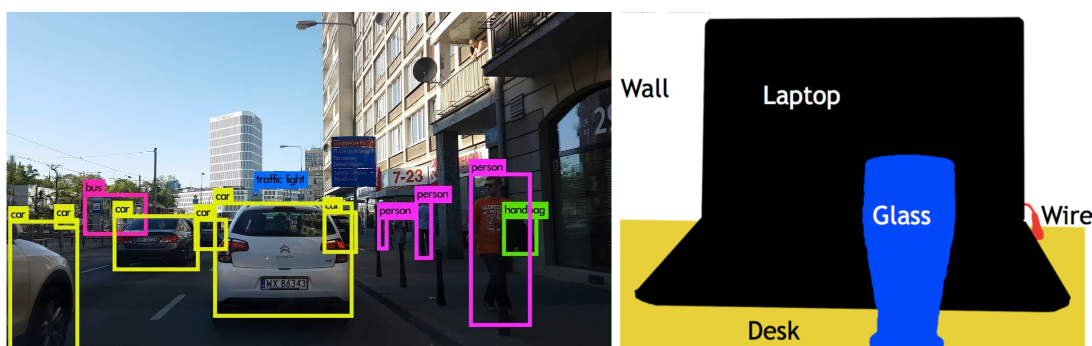


1.1. ábra. *A vizuális informatika főbb ágazatai.*

1.1.1. Alapvető feladatok és nehézségek

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

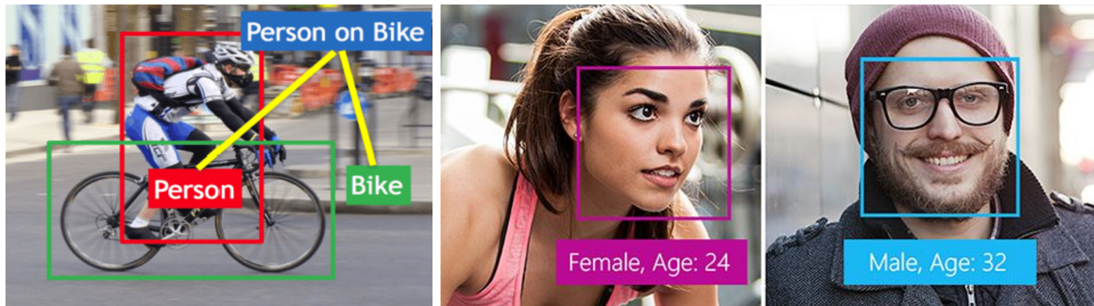
Előfordulhat, hogy az objektumok helyzetén felül annak formájáról és pózáról is szükséges információkat gyűjtenünk, amelyre az objektumokat befoglaló téglalap nem megfelelő. Ekkor célszerű lehet a kép minden egyes pixelét külön osztályozni, így egy olyan maszk képet előállítani, ahol az egyes pixelek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



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

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

döntéseket lehetünk képesek hozni vizuális információk alapján. Ezt a feladatot hívják jelenet értelmezésnek.



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

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

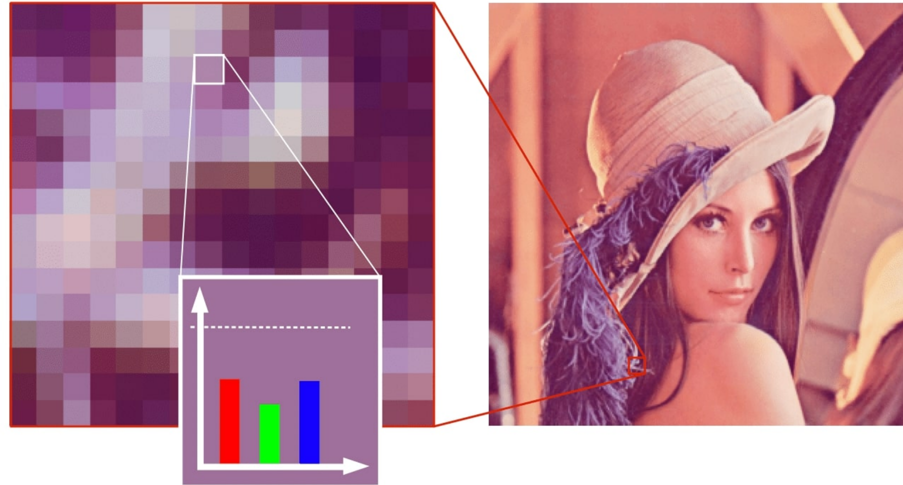
Az osztályozási és detektálási feladat során azonban nem egyetlen konkrét objektumot, hanem egy szemantikus osztályt szeretnénk felismerni. Egy osztályba számtalan különböző objektum tartozhat, amelyek között (az adott osztálytól függően) jelentős eltérések lehetnek. Sőt, a valós világban gyakoriak az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

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

1.2. Képfeldolgozás

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

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



1.4. ábra. *Egy kép felépítése.*

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

1.3. Képi zajok, zajtípusok

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

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

1.3.1. Konvolúciós szűrések

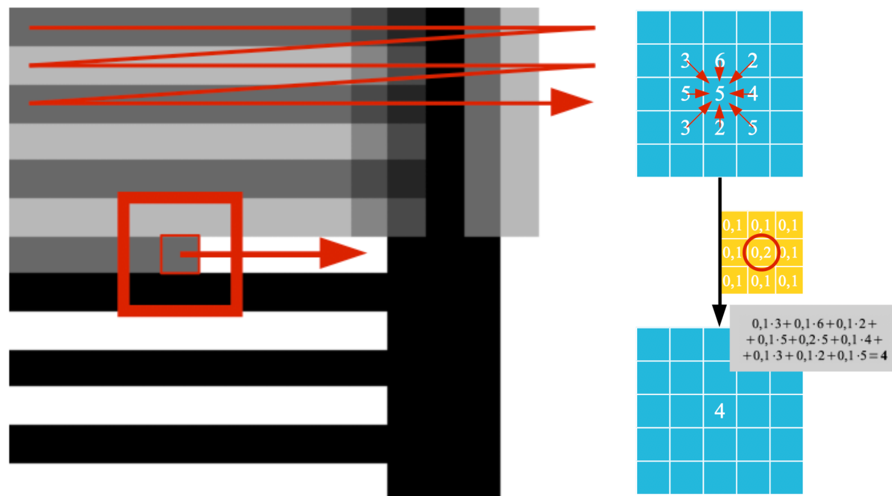
A képjavító algoritmusok családjának legfontosabb tagjai a különböző szűrő algoritmusok, amelyek a képi hibákat és zajokat hivatottak javítani. Ezek az eljárások konvolúciós szűrésen alapszanak. A képen egy kisméretű szűrőablakkal végighaladva, minden egyes pixelpozícióban az adott pixel

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

új értéke a szűrőablak és a pixel lokális környezete között elvégzett konvolúció művelet eredménye lesz. A konvolúció műveletét az alábbi képlet adja meg:

$$(k \circledast I)(x, y) = \sum_{u=-n}^{n} \sum_{v=-n}^{n} k(u, v) * I(x - u, y - v) \tag{1.1}$$

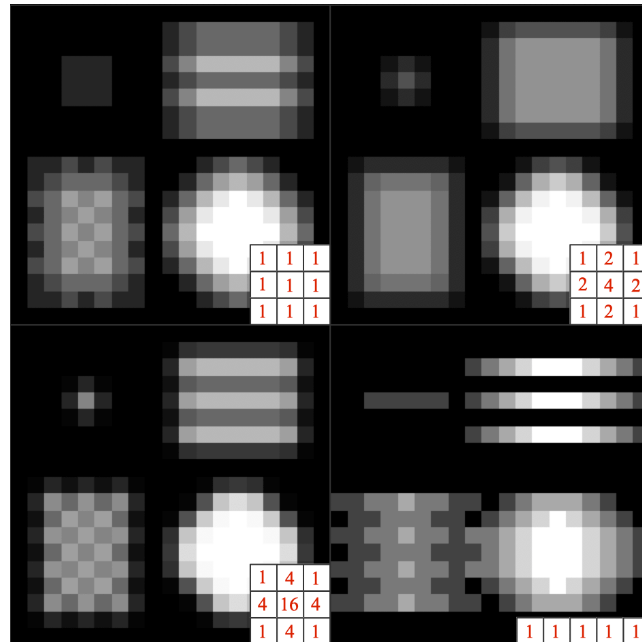
Ahol $I(x, y)$ az y. képsor x. pixele. Mint a képletből is látható, a konvolúció művelete egyszerűen az adott környezetben lévő pixeleknek a szűrőből vett súlyok alapján számított súlyozott összege. A gyakorlatban mindig olyan szűrőket alkalmazunk, amelyeknél a súlyok összege egy, különben a képen világosabbá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



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

Lineáris szűrők

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



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

A simító szűrők egyik legalapvetőbb tulajdonsága, hogy a konvolúciós ablak minden eleme nemnegatív, az eleminek összege pedig pontosan 1. Amennyiben a súlyok összege ettől eltér, a szűrő a simításon felül még világosítja, vagy éppenséggel sötétíti a képet. A konvolúciós szűrők egyik jó tulajdonsága, hogy lineáris műveletek, így egymás utáni szűrések könnyedén összevonhatók.

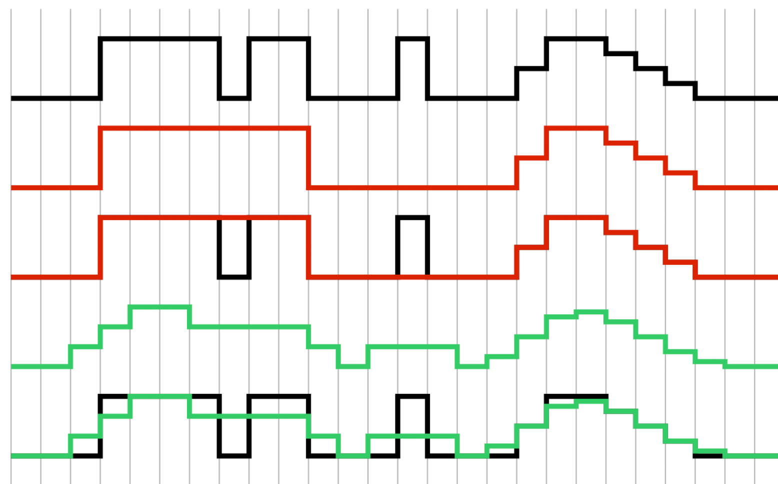
A konvolúciós simító szűrőknek két problémája van: az egyik, hogy az átlagolás után a zajokat ugyan hatékonyan eltüntetik, azonban a szűrés közben a kép egyes részleteit (főleg az éles váltásokat, éleket) is elmossák, ezzel homályossá téve a képet. Másrészt, mivel az összes ilyen szűrő valamilyen átlagolást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

Rang szűrők

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

A mediánszűrők az adott pixelt a környezetükben lévő összes pixel intenzitása közül annak mediánjával, vagyis sorba rendezés után a középső értékkel helyettesítik. E szűrés rendkívüli előnye az, hogy az éles határvonalakat, éleket érintetlenül hagyja, míg rendkívül jól szűri a só és bors

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



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

típusú zajokat. Ennek oka, hogy a mediánstatisztika az átlaggal szemben rendkívül robusztus a ritka, kiugró értékekre. A rangszűrők hátránya, hogy a sorba rendezés művelete rendkívül drága a konvolúcióhoz képest, így ezek a szűrők lényegesen lassabb működést eredményeznek. A helyzetet tovább rontja, hogy néhány magasszintű gyorsítási technika (szeparálható szűrők, frekvenciatartománybeli feldolgozás) is csak konvolúciós szűrőkkel végezhető el.

1.3.2. Élesítés, élkeresés

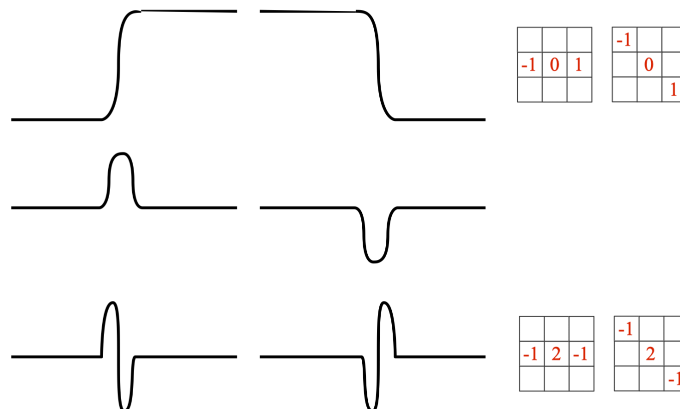
A képi élek definíció szerint a képen található szomszédos pixelek között végbemenő nagymértékű, egyirányú intenzitásváltozások. Lényeges tulajdonságuk, hogy az intenzitás csak az egyik irányban változik, míg a másikban konstans, valamint, hogy a változás éles, ugrásszerű. A valóságban természetesen a különböző képi hibák, zajok és a véges felbontás miatt a fent leírt ideális élekhez képest a valóságban az átmenet fokozatos, elmosott lesz, valamint lokálisan más irányú változás is elképzelhető.

Derivált alapú élkeresés

A képi élek keresésének legegyszerűbb módja a pixelek egyes irányok szerinti deriváltjának számolása, amelyet a konvolúciós szűréshez hasonló elven tehetünk meg numerikusan. A képen végigha-

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

ladva minden pozícióban kiszámítjuk az adott pixel és a jobb, illetve alsó szomszédja különbségét, megkapva a kép x, illetve y irányú deriváltjait. A kettő négyzetes összegéből megkapható a teljes derivált nagysága, amely az adott pont élszerűségének mértékeként értelmezhető.

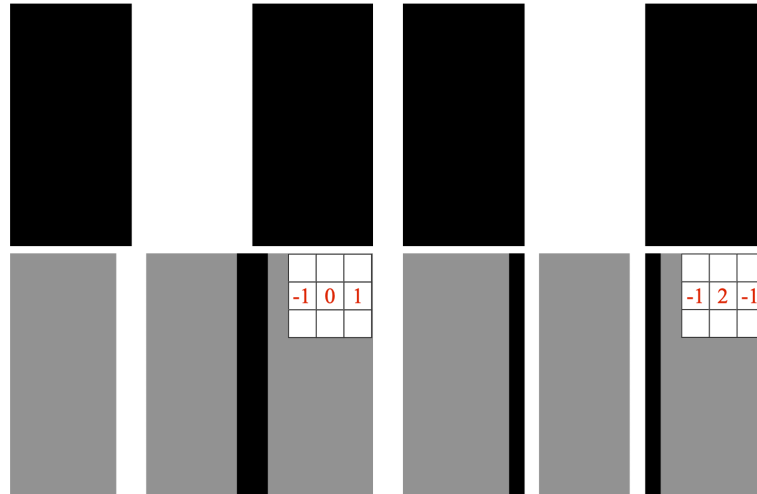


1.12. ábra. *A deriváló szűrők alkalmazása egy képre.*

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

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

Az első deriválton alapuló szűrők egyik legnagyobb hátránya, hogy a simítás miatt az él homályosan, elkenődve fog látszani, így annak pontos lokalizációja nehézkes. Ez a probléma kiküszöbölhető, ha az élképet még egyszer deriváljuk, és a deriváltak nullátmenetének pontját keressük meg. Ezt a műveletet természetesen egyetlen lépésben végezzük el egy második derivált konvolúciós szűrő



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

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

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

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

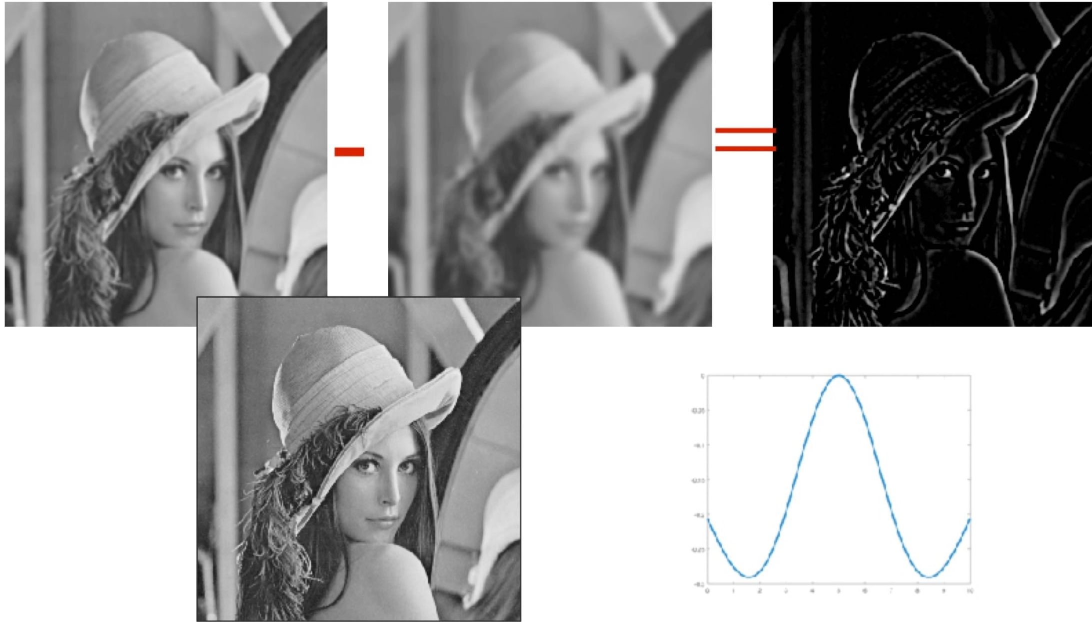
segítségével, amelyet Laplace-szűrőnek nevezünk. A Laplace-szűrőt gyakorta szokták alakja miatt sombrerokalap-szűrőnek is nevezni.

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

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

1.15. ábra. *A Laplace szűrő 4 (bal) és 8 (jobb) szoszédos változata.*

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



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

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

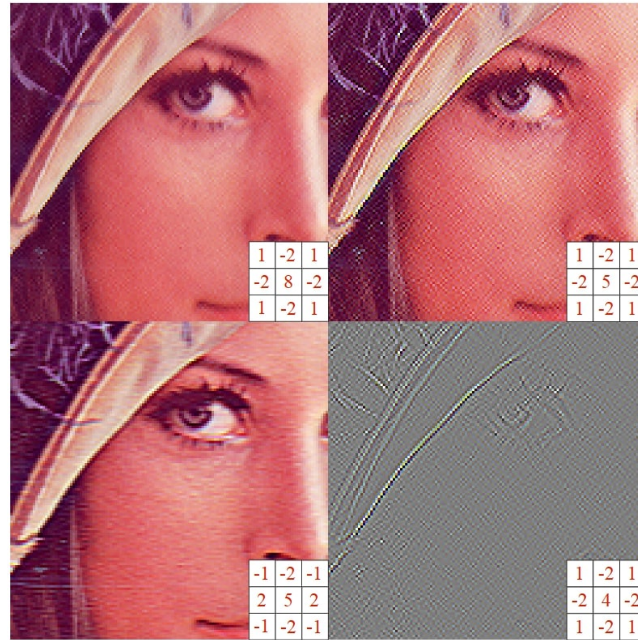
Élesítő szűrők

Megfigyelhető, hogy szemben a simító szűrőkkel, ahol minden szűrő elemeinek összege 1 volt, itt minden élkereső szűrő elemeinek összege 0. Léteznek olyan szűrők is, amelyek, habár értékeik elrendezésében inkább az élkereső szűrőkre hasonlítanak (negatív és pozitív értékek más-más oldalon), mégis az értékeik összege 1. Ezeket élesítő szűrőknek nevezzük, és – mint azt nevük is sugallja – képesek a képeken a finom részleteket, változásokat kiemelni, ezzel a képet élesebb érzetűvé tenni. Ezt a szűrőfajtát gyakorta alkalmazzák fotós alkalmazásokban.

1.4. Template matching

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

A sablonillesztés során alapvetően kétféle illeszkedési függvényt használnak a gyakorlatban. Ezek közül az egyik a sablon és a képrészlet pixelei közötti négyzetes eltérése összege vagy más néven az L2 távolság, amelynek a minimumpontjait keressük. Érdekesebb megoldás azonban a konvolúciós/korrelációs illeszkedés, ahol a sablon és a kép közötti konvolúciót használjuk mérceként. A konvolúció alapvető tulajdonsága, hogy az eredménye akkor lesz abszolút értékben nagy, ha a szűrő



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

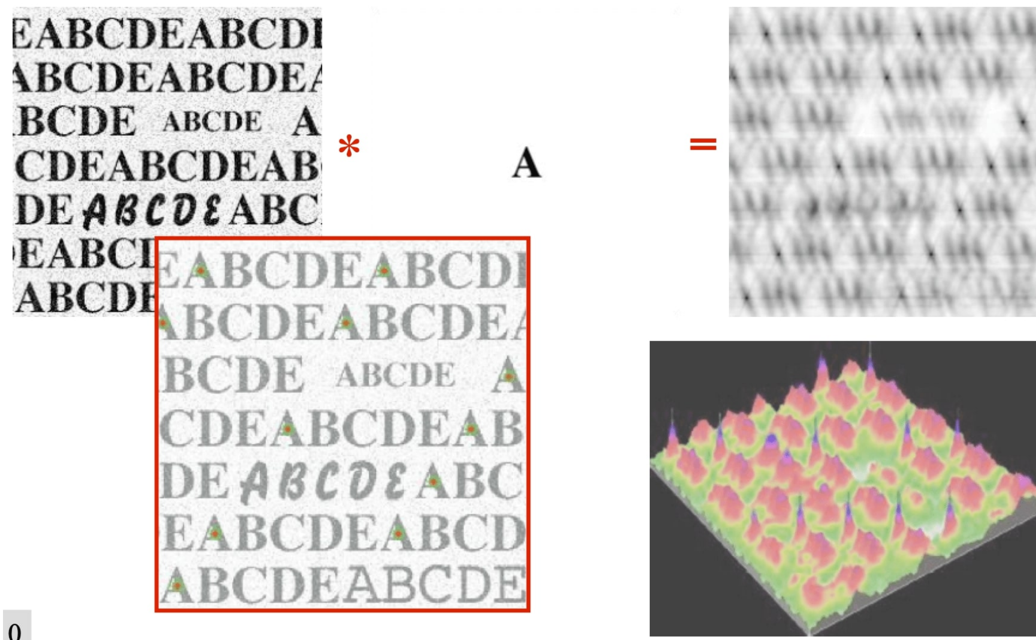
és az általa lefedett képrészletre vagy annak inverzére hasonlít. Ennek illusztrálására említhetjük a korábbi előadáson bemutatott éldetektáló operátorokat, amelyek valóban úgy néznek ki, mint egy képi él.

$$
\begin{aligned}
E_{L2}(x,y) &= \sum_{x'}\sum_{y'}(I(x+x',y+y') - T(x',y'))^2 \\
E_{CC}(x,y) &= \sum_{x'}\sum_{y'} I(x+x',y+y')T(x',y')
\end{aligned}
\tag{1.2}
$$

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

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

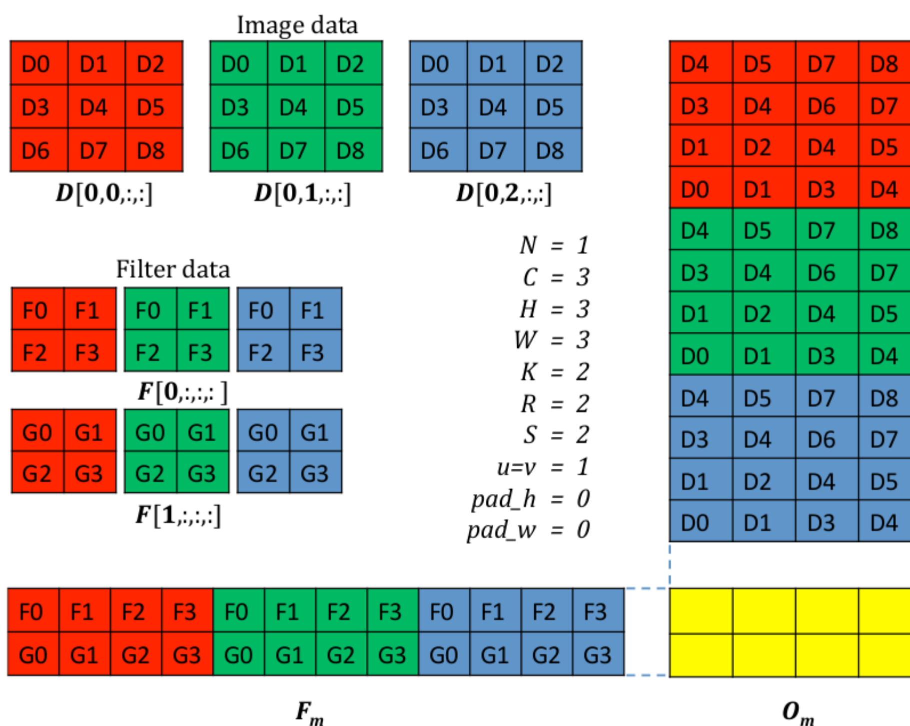
Fontos alkalmazás még a virtuális és kiterjesztett valóság tudományterülete, amelyben gyakorta valósítanak meg beviteli eszközöket oly módon, hogy az eszközökön valamilyen speciális (általában fekete-fehér) jelölő mintázatot helyeznek el. Ezeket a mintákat úgy szokták megválasztani, hogy azok a valóságos objektumokon ne fordulhassanak elő, így e markerek lokalizációja a mintaillesztés segítségével egyszerűen és robusztusan megoldható. Ezt a módszert előszeretettel használják tapintható kiterjesztett valóság rendszerekben, amelyek alapvető elve, hogy a virtuális környezettel való interakciót speciális valós objektumok segítségével valósítják meg.



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

1.5. Hatékony konvolúció

Érdemes emlékezni, hogy a konvolúció lineáris művelet, és mint ilyen, leírható egy mátrixszorzás segítségével. Ehhez azonban a szűrő elemeit és a kép pixeleit az alábbi módon át kell rendeznünk.



1.19. ábra. *A 2D konvolúció, mint mátrixszorzás.*

Mivel az átrendezés művelete relatíve rendkívül alacsony, ezért ez a végrehajtást nem lassítja

számottevően. A mátrixszorzás elvégzésére azonban számos, rendkívül jól optimalizált eljárás létezik, így ezzel a lépéssel a szűrés rendkívül hatékonnyá tehető.

Ezen túl bizonyos szűrőablakok esetén lehetőség nyílik a szűrő szeparálására: ekkor a 2D szűrést, két egymás után 1D szűréssel helyettesíthetjük. Ebben az esetben a szűrő végrehajtásának számítása N^2 helyett $2 \times N$ műveletbe kerül. Ezt azonban csak egy rangú szűrőmátrixok esetén tehetjük meg.

1	2	1

1
2
1

1	2	1
2	4	2
1	2	1

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

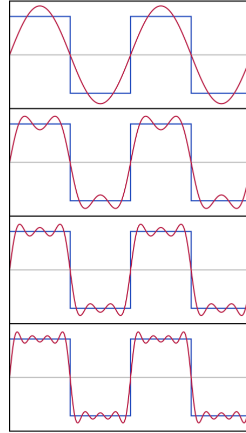
1.5.1. Fourier-transzformáció

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

$$
\begin{gathered}
f(t) = a_0 + \sum_{k=1}^{N} a_k sin(k\omega_0 * t + \phi_k) \\
f(t) = \hat{f}_0 + \sum_{k=-N}^{N} \hat{f}_k e^{i*k\omega_0*t}
\end{gathered}
\tag{1.3}
$$

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

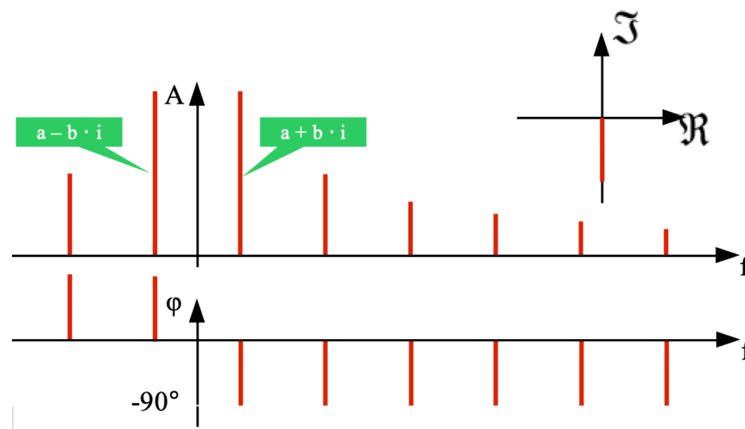
A számítógépes látás tudományterületén szokványos egy kétdimenziós képet a képsík két dimenziójának függvényeként értelmezni. Ebben a vízszintes x és a függőleges y koordináták által kifeszített térben a kép diszkrét pontokban elhelyezkedő, pontszerű impulzusok összességeként írható fel, ahol az egyes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



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

$$F(u,v) = \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} I(x,y) * e^{-i(ux\frac{2\pi}{W}+vy\frac{2\pi}{H})} \tag{1.4}$$

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



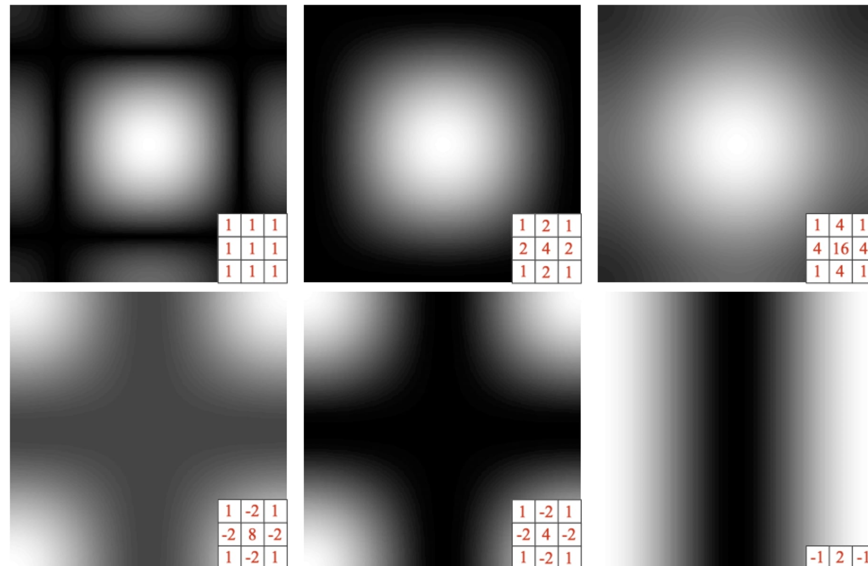
1.22. ábra. *A diszkrét Fourier-transzformáció esetében adódó amplitúdó és fázis spektrumok.*

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

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

Konvolúciós szűrők

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



1.23. ábra. *Egyes konvolúciós szűrők spektruma.*

További Olvasnivaló

[1] Jeff Heaton. „Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning”. *Genetic Programming and Evolvable Machines* 19.1-2 (2017. okt.), 305–307. old. DOI: 10.1007/s10710-017-9314-z. URL: https://doi.org/10.1007%2Fs10710-017-9314-z.

2. fejezet

Neurális hálózatok

2.1. Bevezetés

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

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

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

2.2. Tanuló algoritmusok felépítése

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

$$\hat{y} = f(x, \vartheta) \tag{2.1}$$

Minden tanító algoritmushoz tartozik egy költségfüggvény (gyakran nevezik még hiba- vagy veszteségfüggvénynek), amely a tanuló algoritmus kimentéhez hozzárendel egy hiba értéket, melynek

segítségével az algoritmus teljesítményét tudjuk értékelni. Ezen felül minden tanító algoritmusnak része egy vagy több optimalizálási módszer is, amelynek segítségével a hibafüggvényt minimalizálhatjuk a paraméterek változtatásával. A legtöbb tanító algoritmushoz tartoznak még úgynevezett hiperparaméterek is, melyek olyan paraméterek, amelyek a megoldás minőségét általában befolyásolják, azonban nem tudjuk őket az optimalizálási módszerrel meghatározni. Tipikusan magának az optimalizálási módszernek, vagy a modell struktúrájának tulajdonságai ilyenek.

2.2.1. Tanulás típusai

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

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

Létezik azonban felügyelet nélküli tanulás, amikor a tanító adathalmaz csak bemeneti értékekből áll, az elvárt kimenetet egyáltalán nem ismerjük. Ilyen esetekben azt várjuk el a tanuló algoritmustól, hogy képes legyen valamilyen belső struktúrát találni az adathalmazban, és ezáltal azt kompakt módon leírni. A gépi tanulás harmadik fő fajtája a megerősítéses tanulás, ami két fontos dologban különbözik a másik két típustól. Egyrészt a megerősítéses tanulás esetén szinte mindig összefüggő döntések sorát kell az algoritmusnak meghozni, de a döntéssorozat helyességéről jellemzően nem kap minden döntés után visszajelzést. Másfelől a kapott visszajelzés során az algoritmus csak egy értékelést kap a döntések minőségéről, azt nem tudja meg, hogy a helyes döntés mi lett volna. A megerősítéses tanulási feladatokra tipikusan jó példák a különböző játékok (pl. sakk, go, számítógépes játékok) valamint a különböző járműirányítási feladatok.

2.2.2. Nehézségek

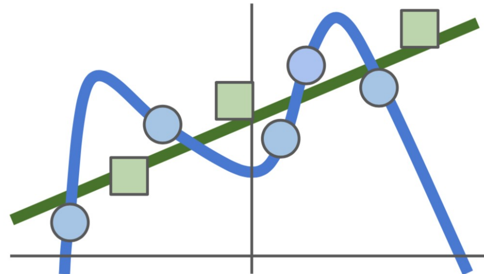
Első ránézésre a gépi tanulás könnyedén tűnhet egyfajta varázslatos módszernek, amivel a világ összes problémáját könnyedén meg lehet oldani. A gyakorlatban azonban ezeknek a módszereknek is bőségesen akadnak limitációik és csapdáik, amikbe könnyedén bele lehet esni. A csapdák elkerülésének érdekében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

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

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

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

A túltanulás oka az, hogy a tanításra használt adathalmaz nem teljesen tökéletes. Egyrészt véges, ami azt jelenti, hogy az algoritmus feladata, hogy megtanuljon általánosítani az adathalmaz segítségével. Másrészt mind a bemenetek, mind a kimenetek zajjal terheltek, így az algoritmus tanítási hibája tökéletes általánosítás esetén sem lesz nulla. Ez azt jelenti, hogy egy idő után az algoritmus már csak úgy tudja tovább csökkenteni a tanítási hibát, ha elkezdi egyesével memorizálni a tanító adatokra adandó helyes választ. Ennek következtében egy a problémát általánosságban megoldó algoritmus helyett egyre inkább egy asszociatív memóriára (9. ábra) kezd hasonlítani. Ez azt jelenti, hogy a tanító adatbázisban nem szereplő bemenetekre egyre rosszabb válaszokat ad. Ráadásul minél komplexebb az algoritmus, annál könnyebben tud egy nagy adatbázist memorizálni.



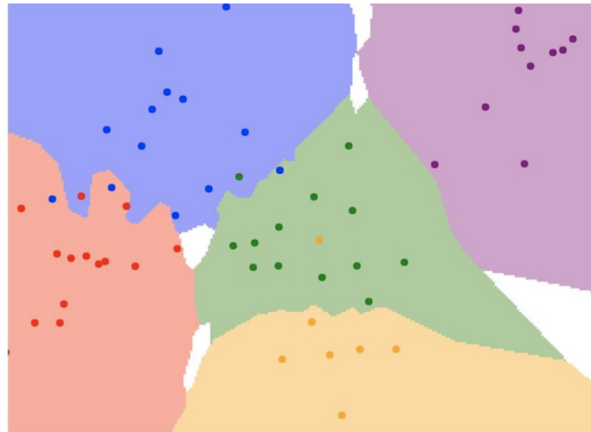
2.1. ábra. *Az overfitting egy dimenzió esetén. Látható, hogy a kék színű modell rátanul a tanító adatbázisban lévő zajra, így a tesztadatokon (zöld) rosszul teljesít.*

2.3. Képosztályozás

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

2.3.1. Legközelebbi szomszéd

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



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

A megoldás alapvető problémája, hogy az intenzitás és színértékek közti különbségek összege nincs összhangban a képek hasonlóságával, különösen nem a szemantikus osztályok közti különbséggel. Könnyen belátható, hogy a különböző megvilágítással vagy háttér előtt készült képek különbsége jelentős lesz a rajtuk szereplő objektum osztályától függetlenül. A helyzet különösen rossz olyan objektumok esetén, amelyek hajlamosak számos különböző színezetben előfordulni (például állatok, járművek, ember). E probléma miatt a színinformációt csak olyan esetekben használjuk képek osztályozására, amikor mind az objektumok kinézetét, mind a környezet vizuális tulajdonságait kézben tudjuk tartani. Ilyen szituációkra jó példák a különböző beltéri ipari alkalmazások (például alkatrészfelismerés) vagy a virtuális- és kiterjesztettvalóság-rendszerek.

2.3.2. Lineáris osztályozás

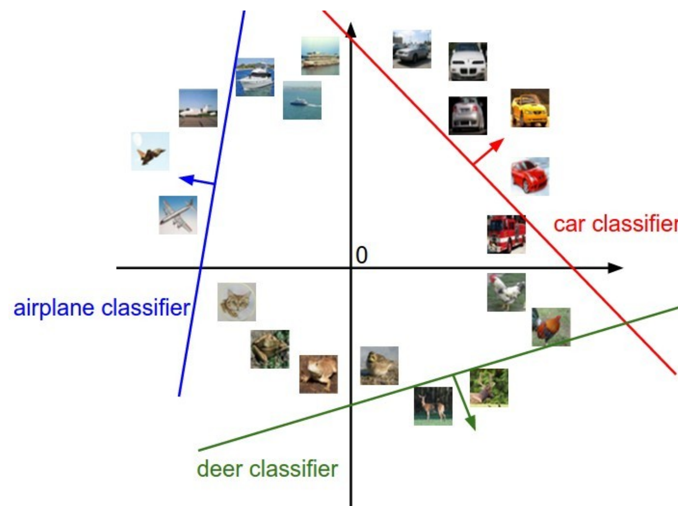
A felügyelt tanulás alapvető módszereinek rövid bemutatása után a jelenlegi fejezettel kezdődően a mély tanulás alapú látórendszerek területének módszereit fogom részletesen bemutatni. A mély tanuló rendszerek alapeleme egy lineáris osztályozó algoritmus, amelynek számos elnevezése létezik. Gyakran szokás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

$$s = Wx \tag{2.2}$$

Ahol x a bemenet, s az osztály jóság, W pedig a paraméterek, vagy súlyok mátrixa (ezt a jelölésrendszert a jelen fejezetben következetesen alkalmazzuk). Az osztályozás ilyen módját úgy lehet elképzelni, hogy a W mátrix i-edik sora kijelöl egy olyan irányt a pixelek terében, amerre az i-edik

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

osztály jósága nő. Ennek alapján az egyes osztályok közötti döntési határok egyenes szakaszokból tevődnek össze (bináris esetben egyetlen egyenes/hipersík).



2.4. ábra. *A lineáris osztályozás esetében az egyes osztályok jósági értékei a tér egy irányában lineárisan nőnek, míg a többiben konstasok. A növekedés irányát a súlymátrix adott osztályhoz tartozó sorvektora határozza meg.*

2.4. A Perceptron tanítása

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

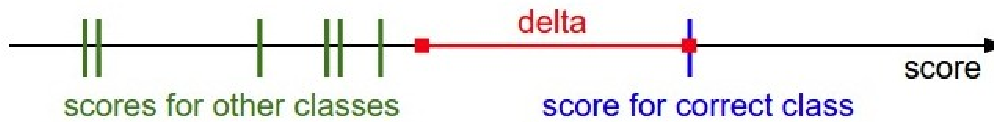
2.4.1. Hibafüggvények

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

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

$$Li = \sum_{j \neq y_i} max(0, s_j - s_{corr} + \Delta) \tag{2.3}$$

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



2.5. ábra. *A Hinge költség szemléletesen.*

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

$$H(p) = -\sum_i p_i log p_i \tag{2.4}$$

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

$$H(p, q) = -\sum_i p_i log q_i \tag{2.5}$$

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

$$q_{k,i} = q(y_k|x_i) = \frac{e_{s_k}}{\sum_j e_{s_j}} \tag{2.6}$$

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

$$\begin{aligned} L_i = H(p_i, q_i) = -\sum_k p_{k,i} log q_{k,i} \\ L_i = -log q_{true,i} \end{aligned} \tag{2.7}$$

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

2.4.2. Regularizáció

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

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

$$\begin{aligned} L = \sum_i^N L_i + \lambda R(W) \\ R_{L1}(W) = \sum_k \sum_l |W_{k,l}| \\ R_{L2}(W) = \sum_k \sum_l W_{k,l}^2 \\ R_{EL}(W) = \sum_k \sum_l \left(\beta W_{k,l}^2 + |W_{k,l}|\right) \end{aligned} \tag{2.8}$$

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

2.5. Optimalizáció

Az előző előadás egyik legnagyobb lezáratlan kérdése, hogy miként lehet a korábban említett perceptron modell hibafüggvényét minimalizálni. Ebből kifolyólag az első témánk a hibafüggvények

minimalizálására szolgáló optimalizálási módszerek tárgyalása. A perceptron súlyainak optimális értékére nem létezik zárt alakú megoldás, így iteratív optimalizálásra lesz szükség.

2.5.1. Gradiens alapú módszerek

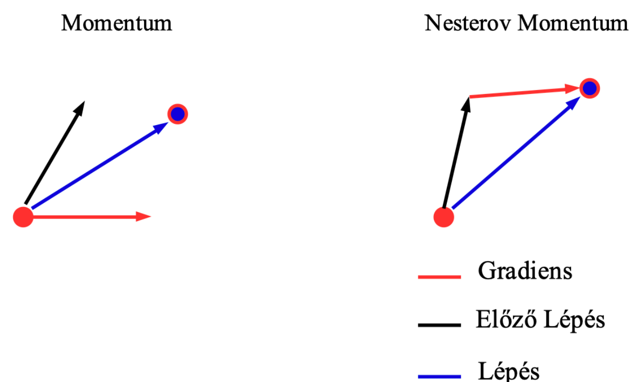
Szerencsére azonban a modell és a költségfüggvény is deriválható, így alkalmazhatunk gradiens alapú módszereket. Ha kiszámoljuk a hibafüggvény súlyok szerinti deriváltját (más szóval a súlyok gradiensét), akkor megkapjuk azt, hogy hogyan kellene a súlyoknak megváltozni ahhoz, hogy a hibafüggvény a lehető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

Vegyünk azonban észre, hogy mivel a teljes hibát szeretnénk minimalizálni, ezért minden egyes lépéskor ki kell az értékelni az egész tanító adatbázis hibáját. Mivel a gradiens módszer aránylag sok lépés után konvergál csak, azért ez nem praktikus. Éppen ezért a tanító adatbázist egyenlő méretű, véletlenszerűen kiválasztott részhalmazokra (minibatch-ekre) osztjuk, és minden egyes minibatch után végzünk egy lépést. Ezt a módszert sztochasztikus gradiens módszernek (SGD - Stochastic Gradient Descent) nevezzük. (Ruder, 2016) A minibatchek mérete általában a kettő hatványa szokott lenni, implementációs okokból. Érezhető persze, hogy az egyetlen minibatch-re számolt gradiens nem egyezik teljesen meg az egész adatbázisra számolttal, de elég közel van ahhoz, hogy megközelítőleg a jó irányba haladjon a módszer. Mivel így egyetlen lépést sokkal olcsóbb végrehajtani, összességében jelentős gyorsulást érünk el. A gradiens módszer képlete a következő:

$$W_{k+1} = W_k - \alpha \frac{\partial ||E||^2}{\partial W} \tag{2.9}$$

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

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



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

A sztochasztikus gradiens módszer egyik hiányossága, hogy a sokdimenziós paramétertér minden irányát egyenértékűnek veszi. Előfordulhat azonban, hogy a költségfüggvény az egyik irányban

meglehetősen meredek, így ebbe az irányba óvatoson érdemes haladni, mivel egy nagyobb ugrással könnyen átugorhatjuk a minimum helyét. Eközben egy másik irányban a költségfüggvény lapos, az optimum pedig meglehetősen messze található. Míg az első probléma a lépésméret csökkentését igényelné, a második esetben pont növelni kellene, a kettőt egyszerre pedig nem lehet. További problémája ezeknek a módszereknek, hogy a hiperparaméterek értékére rendkívül érzékenyek. Ezeknek a beállítása általában számos egymást követő tanítást igényel.

2.5.2. Adaptív momentum (Adam) módszer

Ezekre a problémákra kíván megoldást adni az Adam algoritmus, amely más, a jelen tárgy keretében részletesen nem tárgyalt módszerek (AdaGrad és az RMSProp módszerek) kombinációjának tekinthető. Az Adam módszerben két alapvető ötlet kombinációját alkalmazzuk: az első az úgynevezett adaptív momentum számítás, amely a hagyományos, illetve a Nesterov momentum mellett egy harmadik számítási módszer. Lényege, hogy a momentum aktuális értéke az előző momentum érték és az aktuális gradiens érték súlyozott átlaga. Ezzel tulajdonképpen egy exponenciális átlagolást valósítunk meg az alábbi módon:

$$M_k = \beta_1 M_{k-1} + (1 - \beta_1)\nabla L \tag{2.10}$$

A másik ötlet az adaptív gradiens skálázás, melynek segítségével az Adam algoritmus megjegyzi, hogy egyes irányokban mekkorák voltak a múltbéli deriváltak, majd a ”meredekebb” irányokba óvatosabban, a ”laposabb” irányokba pedig bátrabban lép. Ehhez az algoritmus nem csak a gradiensekből képez exponenciális átlagok, hanem azok négyzetéből is, melynek eredményeképp előáll egy mennyiség, ami pont azt fejezi ki, hogy az egyes gradiens komponensek általában milyen nagyok.

$$G_k = \beta_2 G_{k-1} + (1 - \beta_2)||\nabla L||^2 \tag{2.11}$$

Az Adam algoritmus végső lépési szabálya ezen mennyiségek segítségével fejezhető ki:

$$W_{k+1} = W_k - \alpha \frac{M_k}{G_k}, \tag{2.12}$$

ahol a tört a számlálóban és a nevezőben szereplő vektorok közti elemenkénti osztást jelöli. Vagyis, az algoritmus mindig az adaptívan számolt momentum irányába lép (vegyük észre, hogy ebben már benne van az aktuális gradiens), azonban ez az átlagos gradiens nagyságokkal le van skálázva.

Az Adam algoritmus az egyik leginkább elterjedt optimalizálási módszer, azonban van egy súlyos hátránya: Az optimalizálás kezdeti lépéseinél a G_k és M_k értékek csupán néhány elem átlagából adódnak, így az értékük rendkívül instabil. Ennek eredményeképp az Adam módszer kezdeti lépései teljesen rosszak is lehetnek, melynek eredményeképp az algoritmus beragadhat egy lokális minimumba, amelyből később már egyáltalán nem képes kimászni.

Ennek gyakori elkerülési módja a Warmup, melynek során az Adam módszert nagyon kicsi tanulási rátával egy egyész epochon keresztül futtatják, és az igazi optimalizálást csak ezután indítják el. Egy relatíve új változat, az ún. Rectified Adam (RAdam) módszer teljesen képes megoldani ezt a problémát egy kompenzáló szorzó tényező bevezetésével.

2.5.3. Másodrendű módszerek

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

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

$$f(W_k + \Delta W) \approx f(W_k) + f'(W_k)\Delta W + \frac{1}{2}f''(W_k)\Delta W^2 \tag{2.13}$$

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

$$\begin{aligned} 0 &= f'(W_k) + f''(W_k)\Delta W \\ \Delta W &= -\frac{f'(W_k)}{f''(W_k)} \\ W_{k+1} &= W_k - \frac{f'(W_k)}{f''(W_k)} \end{aligned} \tag{2.14}$$

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

$$W_{k+1} = W_k - H_W^{-1} \ \ \nabla_W f(W_k) \tag{2.15}$$

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

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

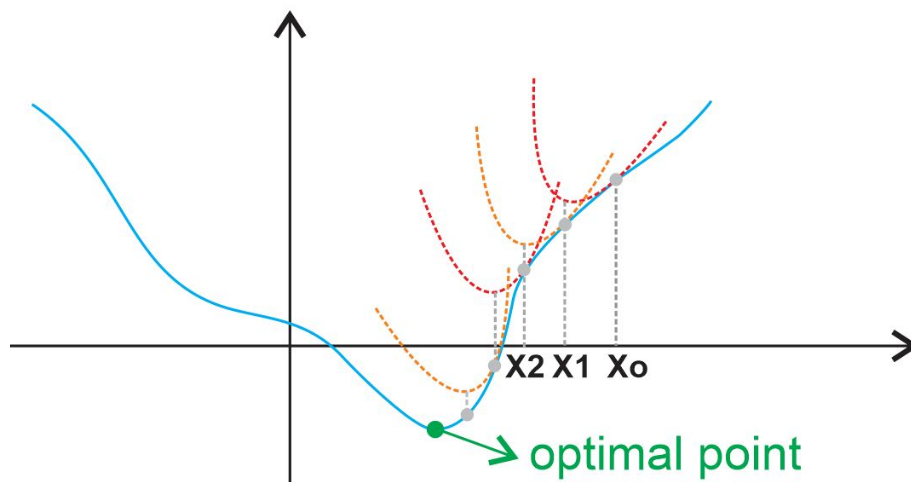
$$W_{k+1} = W_k - [H_W + \lambda I]^{-1} \ \ \nabla_W f(W_k) \tag{2.16}$$

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

2.5.4. Backpropagation

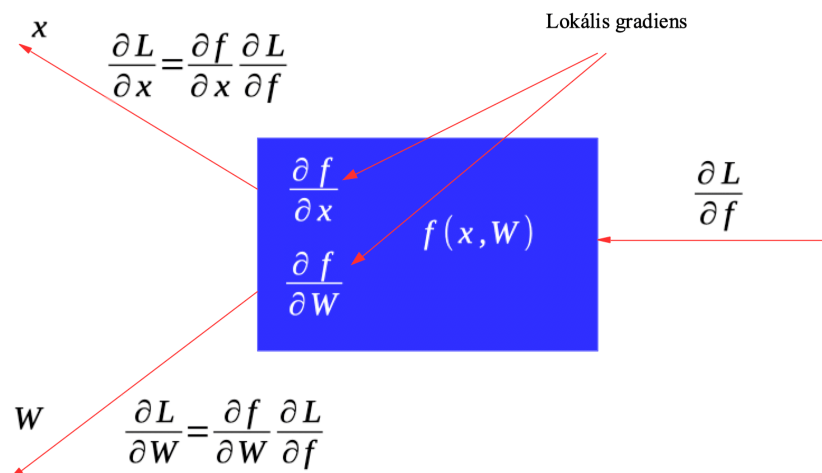
Amint azt már korábban említettük, az egyszerű lineáris modellek képességei erősen limitáltak, így a képi bemenetek esetében ritkán használjuk őket. Ismertetésük azonban szükséges volt, ugyanis ezek az egyszerű lineáris modellek könnyedén összeépíthetők komplex nemlineáris tanuló eljárásokká. Amennyiben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

Az egyetlen kérdéses lépés az a hibafüggvény súlyok szerinti deriváltjának számítása. Egy neurális háló elképzelhető, mint egy számítási gráf, ahol a gráf egyes csomópontjai egyszerű, analitikusan



2.7. ábra. *A Newton algoritmus elve.*

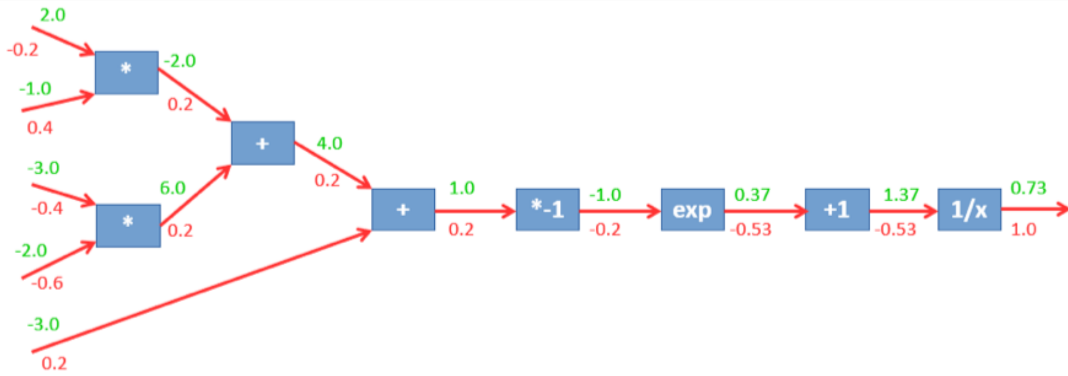
deriválható függvényeket implementálnak. Amennyiben egy számítási gráfban ismerjük a bemeneteket, és az egyes csomópontok által implementált függvényeket, akkor az összes csomópont kimenetét ki tudjuk számítani. Ezt nevezzük az előreterjesztés műveletének. Érdemes azonban észrevenni, hogy amennyiben ismerjük a csomópontok függvényeit, és ismerjük a számunkra érdekes mennyiség (ez esetben a hibafüggvény) deriváltját a csomópont kimenete szerint, akkor a deriválás láncszabályának segítségével könnyedén előállíthatjuk a csomópont bemenete és súlyai szerinti deriváltakat.



2.8. ábra. *A láncszabály szemléltetése.*

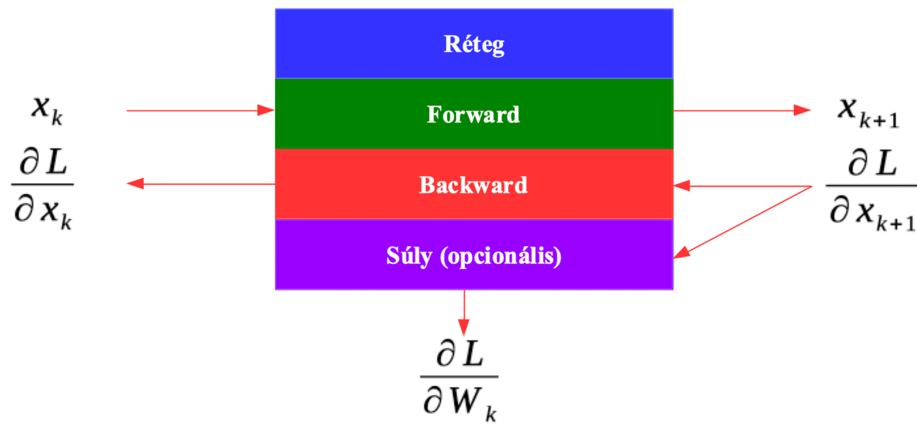
Ahol L a hibafüggvény, f és x az adott réteg ki- és bemenete, W pedig a réteg bemenete és paraméterei. Ezzel a módszerrel a hálón hátrafele haladva az összes bemenet és súly gradiensét meg tudjuk határozni. Ezt a műveletet hátraterjesztésnek (backpropagation) nevezzük. Az egyetlen kérdés csupán, hogy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

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



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

és hátraterjesztés részfeladatait elvégző függvényeket megvalósítjuk.



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

További Olvasnivaló

[1] Jeff Heaton. „Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning”. *Genetic Programming and Evolvable Machines* 19.1-2 (2017. okt.), 305–307. old. DOI: 10.1007/s10710-017-9314-z. URL: https://doi.org/10.1007%2Fs10710-017-9314-z.

[3] Diederik P. Kingma és Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2017. eprint: 1412.6980. URL: http://www.arxiv.org/abs/1412.6980.

[4] Liyuan Liu és tsai. *On the Variance of the Adaptive Learning Rate and Beyond.* 2020. eprint: 1908.03265. URL: http://www.arxiv.org/abs/1908.03265.

[5] Donald W. Marquardt. „An Algorithm for Least-Squares Estimation of Nonlinear Parameters”. *Journal of the Society for Industrial and Applied Mathematics* 11.2 (1963. jún.), 431–441. old. DOI: 10.1137/0111030. URL: https://doi.org/10.1137%2F0111030.

3. fejezet

Konvolúciós Neurális Hálók

3.1. Bevezetés

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

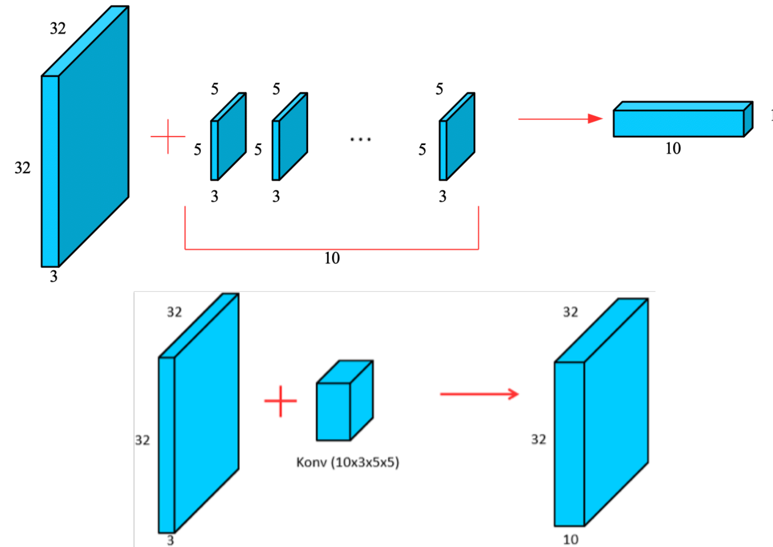
3.2. Konvolúciós neurális hálók

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

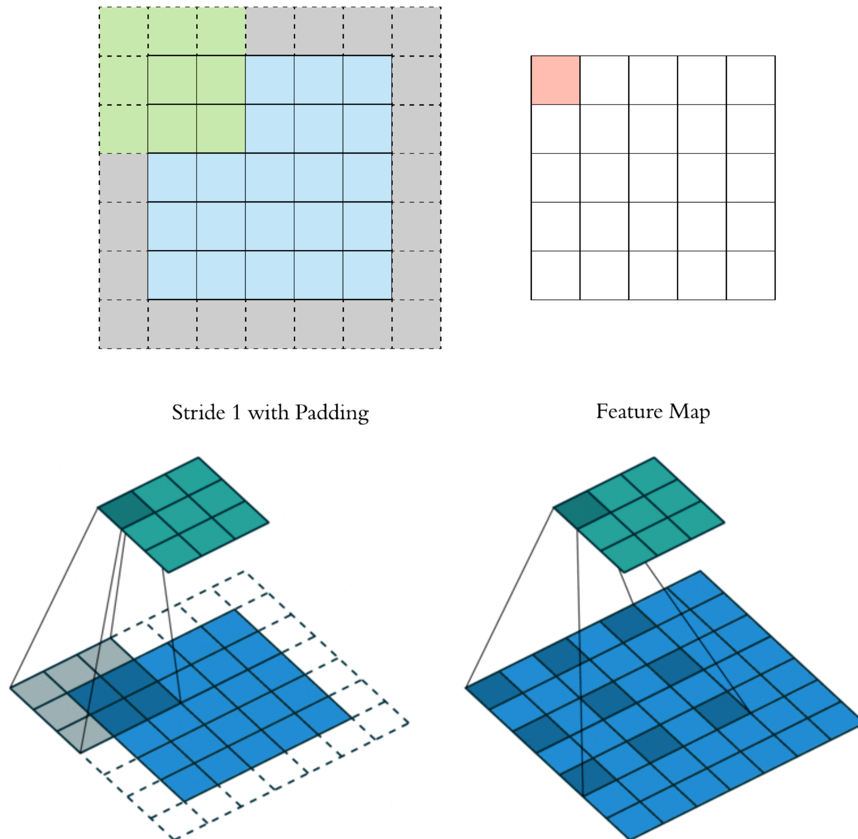
3.2.1. Konvolúciós réteg

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

A konvolúciós rétegeknek még két fontos paramétere létezik: a stride, és a dilatáció. A konvolúciós szűrő egyes stride esetén minden pixelen végighalad, míg kettes stride esetén minden másodikon, és így tovább. Ezt a beállítást a kép térbeli méretének csökkentésére szokták használni. A dilatáció (5.2 ábra) értéke azt határozza meg, hogy a szűrőn eggyel arrébb található súlyt a képen hánnyal arrébb lévő pixellel szorozzuk. Egyes dilatáció értéke esetén a szűrő a megszokott módon viselkedik, egynél nagyobb érték esetén viszont egyre inkább „széthúzódik”. Ezt a beállítást arra szokták használni, hogy a konvolúciós szűrők által érzékelt képrészlet méretét növeljék.



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

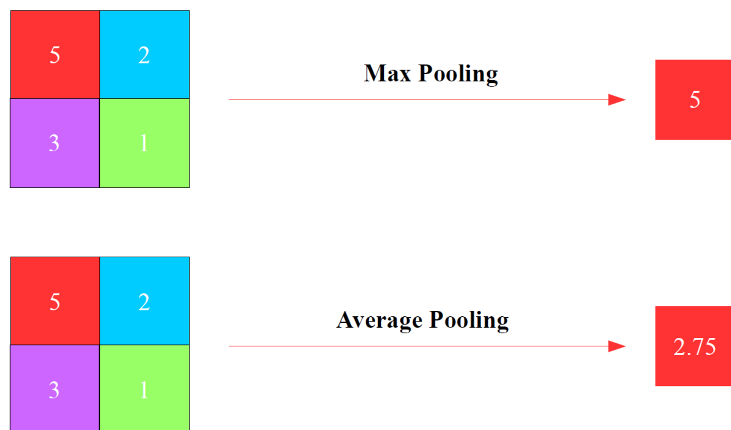


3.2. ábra. *A padding (felül), a stride (alul bal) és a dilatáció (alul jobb) hatása a konvolúció műveletére.*

3.2.2. Pooling

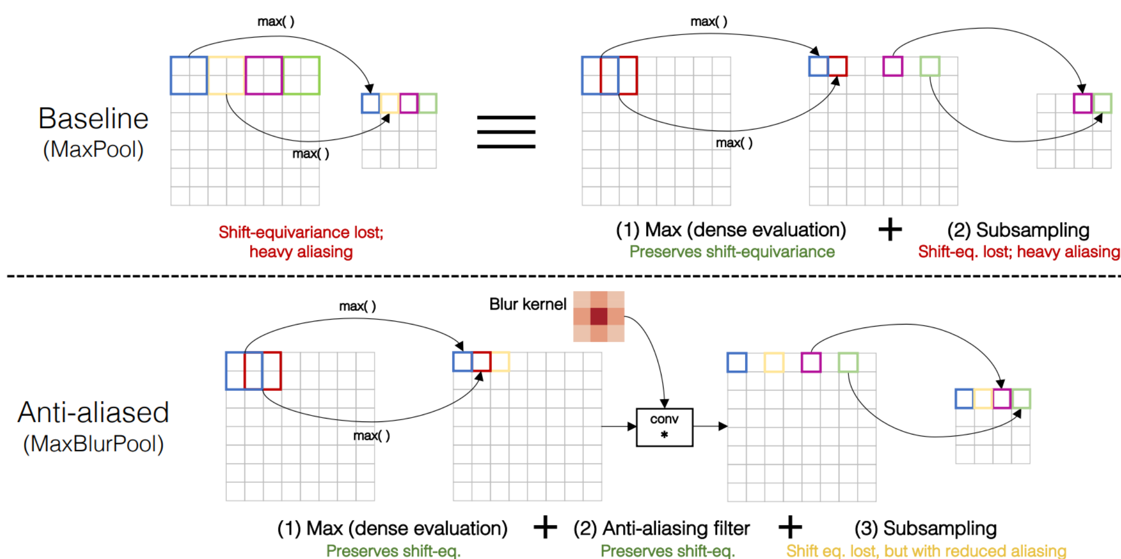
Érdemes észrevenni, hogy amennyiben egymás után újabb és újabb konvolúciós rétegeket helyezünk el, egyre növekvő mélységgel, akkor egy idő után a rétegek kimenetén kapott aktivációs tömb mérete hatalmas lesz. Éppen ezért néhány rétegenként érdemes az aktivációs tömb térbeli méreteit

csökkenteni. Az egynél nagyobb stride paraméterrel rendelkező konvolúciós réteg mellett ezt még az úgynevezett pooling operáció segítségével is megtehetjük. A pooling szintén egy csúszóablakos művelet, amely az aktivációs tömb éppen lefedett részét egyetlen számmal helyettesíti. A két leggyakoribb eset, amikor ez az érték az ablak által lefedett érékek átlaga vagy maximuma. Érdemes megjegyezni, hogy manapság egyre gyakoribb a strided konvolúció használata a pooling helyett.



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

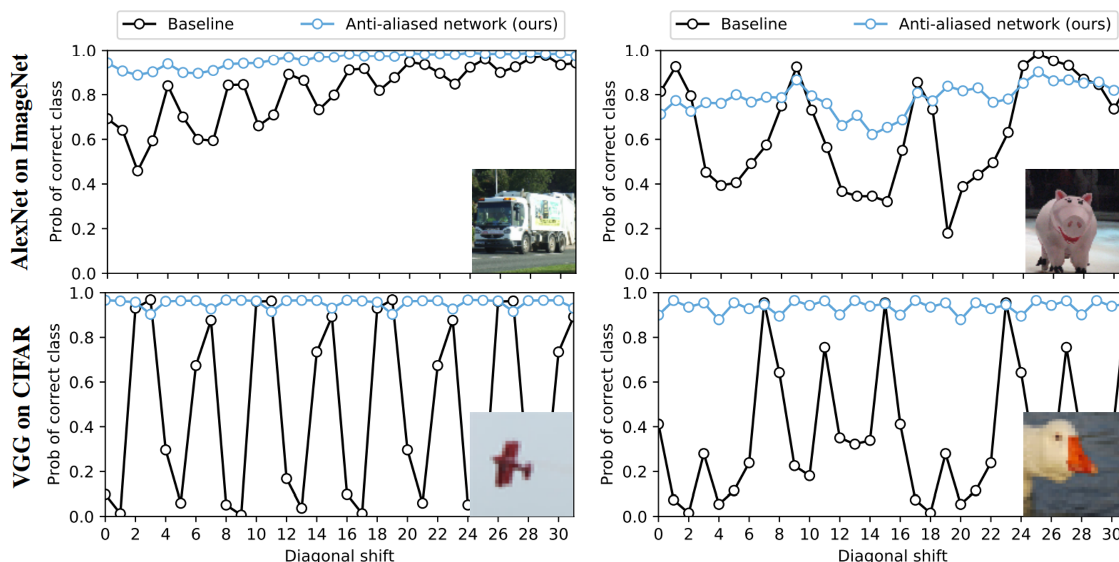
Fontos azonban tudni, hogy az eddig tárgyalt leskálázó módszerek nem invariánsak a transzlációra, így a konvolúciós hálózatoknak ezt a kívánatos tulajdonságát elrontják. Tipikusan képek/más jelek esetén az alulmintavételezés előtt szükséges egy aluláteresztő szűrést végezni, különben 1-2 pixeles eltolások nagy változásokat okozhatnak az alulmintavételezett képben. Ez a technika Anti-aliasing néven ismert, és a számítógépes grafikában széles körben alkalmazzák. Ezt a módszert a pooling operációba a következőképpen lehet beépíteni: Képzeletben bontsuk fel a 2 méretű poolingot két lépésre. Az első lépésben egy 2×2 méretű kernellel végig megyünk a képen (2-es stride értékkel), és minden pixelt lecserélünk a kernelben lévő pixelek maximumával/átlagával. Ekkor a kép térbeli mérete még nem változik. Ezt követően pedig egyszerűen minden második oszlopot és minden második sort megtartva alulmintavételezzük a képet.



3.4. ábra. *A max pooling felbontása (felül) és az Anti-aliased max pooling (alul).*

Az ily módon felbontott pooling rétegbe az aluláteresztő szűrés már egyszerűen beépíthető a két lépés közé. Érdemes megjegyezni, hogy a módszer ugyanúgy beépíthető strided konvolúció használata esetén, hiszen ez a módszer is felfogható egy hagyományos konvolúció és egy stride mértékű alulmintavételezés kombinációjaként. Az így kapott Blur Pooling rétegek nagymértékben javítják

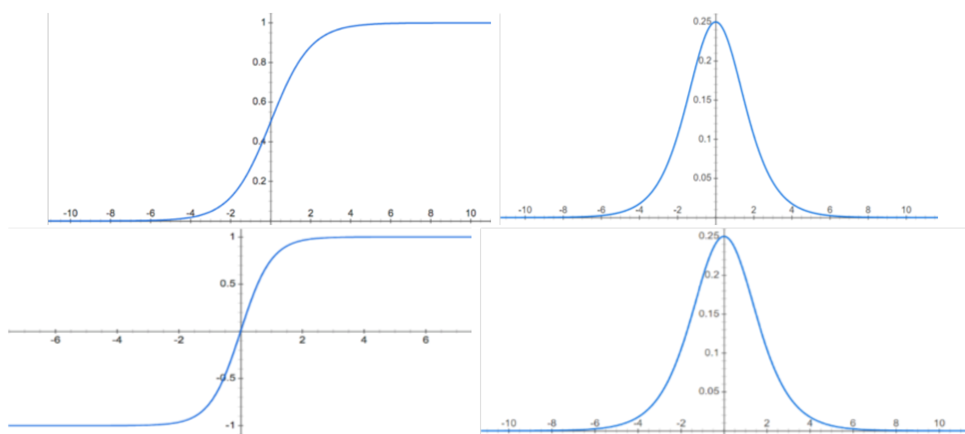
a neurális hálók eltolás invarianciáját.



3.5. ábra. *A blur pooling használatának hatása kis eltolások esetén.*

3.2.3. Aktivációk

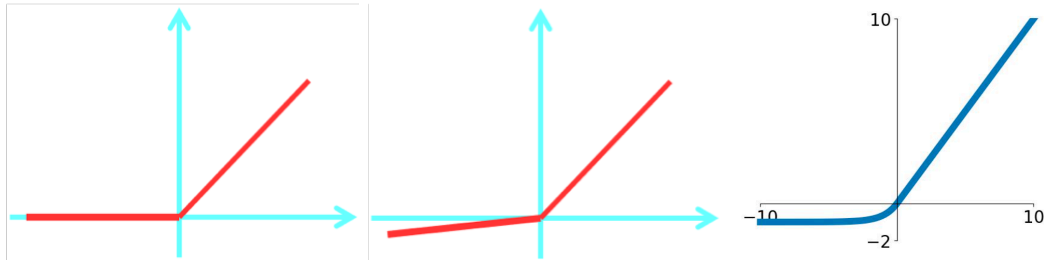
A többrétegű neurális hálók utolsó esszenciális rétege az aktivációs réteg, ami tipikusan minden konvolúciós és lineáris réteget követ. Ez a két réteg ugyanis lineáris műveleteket hajt végre, ezek kompozíciója is lineáris marad. Éppen ezért az egyes rétegek közé nemlineáris függvényeket ékelünk be, amelyek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



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

A jelenlegi hálók esetén a legnépszerűbb aktivációs függvény a ReLU (Rectified Linear Unit), amely egy szakaszosan lineáris aktiváció, amely a negatív bemeneteket kinullázza, míg a pozitív tartományban nem fejt ki hatást. Ennek az aktivációnak a deriváltja a pozitív tartományban 1, a negatívban 0, így a deriváltakra kifejtett zavaró hatása lényegesen kisebb. Ennek ellenére ReLU használata esetén is előfordulhat az elülső rétegek beragadása, amelyre a paraméteres ReLU (PReLU), valamint a szivárgó (Leaky) ReLU adhat megoldást. Ezek annyiban különböznek az

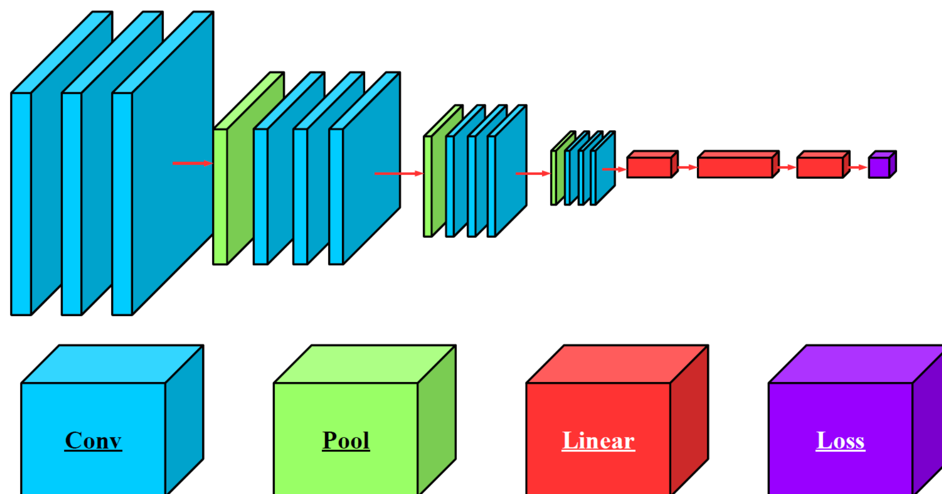
eredeti megoldástól, hogy a negatív tartományban nem nullázzák az aktivációkat, hanem egy egynél kisebb konstanssal szorozzák azokat. A szivárgó esetben ez a konstans egy hiperparaméter, míg a paraméteres esetben a gradiens módszer segítségével tanulható. Létezik továbbá a ReLU egy olyan változata, amely a hagyományos verzióval ellentétben teljesen sima, így minden pontban deriválható. Ezt Elastic Linear Unit-nak (ELU) nevezzük.



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

3.3. Architektúrák

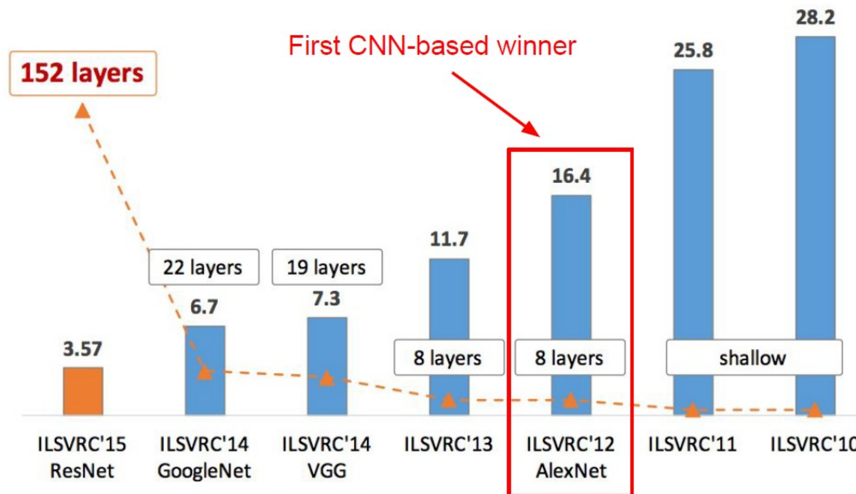
A jelen fejezetben bemutatott rétegeket tartalmazó neurális hálózatokat konvolúciós neurális hálózatoknak nevezzünk, melyeket első sorban a számítógépes látás területén alkalmaznak. A konvolúciós rétegek alkalmazása kifejezetten előnyös képi adatok esetén, mivel egy konvolúciós rétegnek a lineárishoz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



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

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

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



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

3.3.1. AlexNet

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

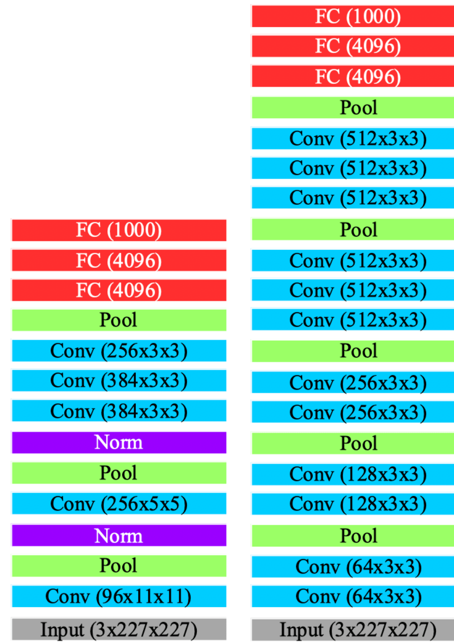
3.3.2. VGG

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

Az alábbi ábrán látható a VGG modell egyes rétegei számára szükséges memória mérete, és a réteg paramétereinek száma. Könnyen észrevehetjük, hogy a háló eleje a háló leginkább memóriaigényes része, paraméterek tekintetében viszont az utolsó rétegek szerepelnek igazán hangsúlyosan. Érdemes megjegyezni, hogy a VGG a modern hálótípusokhoz képest rendkívül pazarló mind paraméterszám, mind memóriahasználat tekintetében.

3.3.3. Inception

Paraméterek csökkentésére irányuló fejlesztésre jó példa az Inception névre hallgató réteg típus. Ennek a megoldásnak a lényege, hogy a konvolúciós rétegek nem csak sorban, hanem egymással



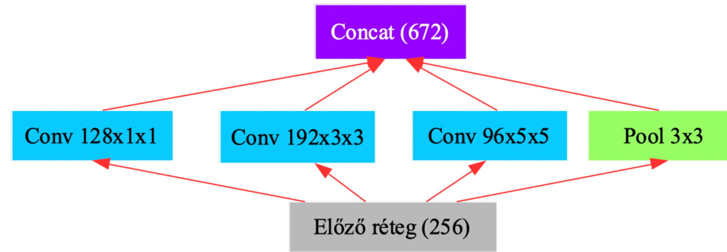
3.10. ábra. *Az AlexNet (bal) és a VGG (jobb) modellek.*

FC (1000)	1000	4M (4096x1000)
FC (4096)	4096	17M (4096x4096)
FC (4096)	4096	102M (7x7x512x4096)
Pool	25k(512x7x7)	0
Conv (512x3x3)	100k (512x14x14)	2.4M (512x512x3x3)
Conv (512x3x3)	100k (512x14x14)	2.4M (512x512x3x3)
Conv (512x3x3)	100k (512x14x14)	2.4M (512x512x3x3)
Pool	100k (512x14x14)	0
Conv (512x3x3)	400k (512x28x28)	2.4M (512x512x3x3)
Conv (512x3x3)	400k (512x28x28)	2.4M (512x512x3x3)
Conv (512x3x3)	400k (512x28x28)	1.2M (256x512x3x3)
Pool	200k (256x28x28)	0
Conv (256x3x3)	800k (256x56x56)	590k (256x256x3x3)
Conv (256x3x3)	800k (256x56x56)	294k (128x256x3x3)
Pool	400k (128x56x56)	0
Conv (128x3x3)	1.6M (128x112x112)	147,456 (128x128x3x3)
Conv (128x3x3)	1.6M (128x112x112)	73,728 (64x128x3x3)
Pool	800k (64x112x112)	0
Conv (64x3x3)	3.2M (64x224x224)	36k (64x64x3x3)
Conv (64x3x3)	3.2M (64x224x224)	1.7k (3x64x3x3)
Input (3x227x227)	Memória ~ 64M	Paraméterek ~ 140M

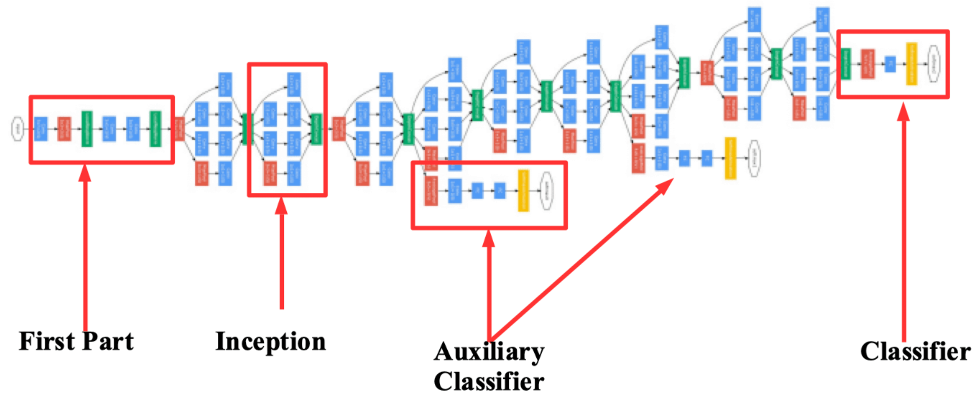
3.11. ábra. *A VGG háló által használt paraméterek száma és memória mérete.*

párhuzamosan is létezhetnek. Ezek a párhuzamos rétegek általában különböző méretű konvolúciókat hajtanak végre, így egy olyan háló struktúrát eredményezve, amely lényegesen jobban tudja kezelni az objektumok skálájában fellépő variációkat.

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



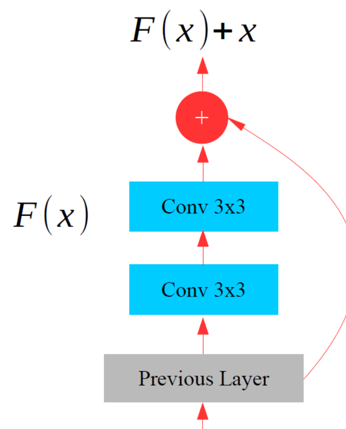
3.12. ábra. *A naiv Inception réteg.*



3.13. ábra. *A GoogLeNet modell.*

3.3.4. ResNet

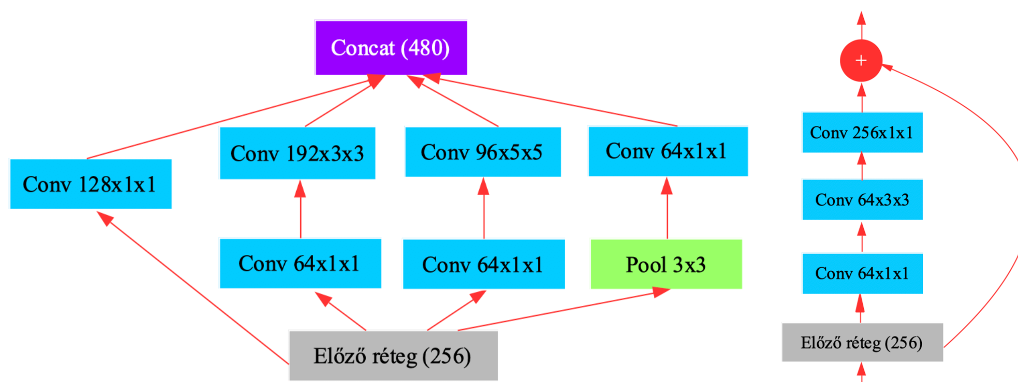
A konvergencia javítására az úgynevezett reziduális blokk jó példa. Ez a réteg a konvolúció végrehajtása utáni kimenethez hozzáadja a bemenetet, így a rétegnek tulajdonképpen az elvárt be- és kimenet közti különbséget kell előállítania. Ennek a megoldásnak az a haszna, hogy az ilyen blokkokból álló neurális hálóban ezeken az összeadásokon keresztül a hiba deriváltja számára egy olyan út vezet vissza a háló elülső rétegeihez, ami mentén a derivált egy. Ennek következtében a hátraterjesztés során végrehajtandó számtalan szorzásból eredő numerikus problémák orvosolhatók. A reziduális blokk bevezetésének hatására a konvolúciós hálók maximális mélysége a 30 réteg körüli értékről a 100-200 réteg nagyságrendjére növekedett.



3.14. ábra. *A reziduális blokk.*

Érdemes még megemlíteni az úgynevezett tömörítő (bottleneck) rétegeket, amiket mind az Inception, mint a reziduális blokkok alkalmaznak. Ezek motivációja az, hogy a kétdimenziós konvolúciók

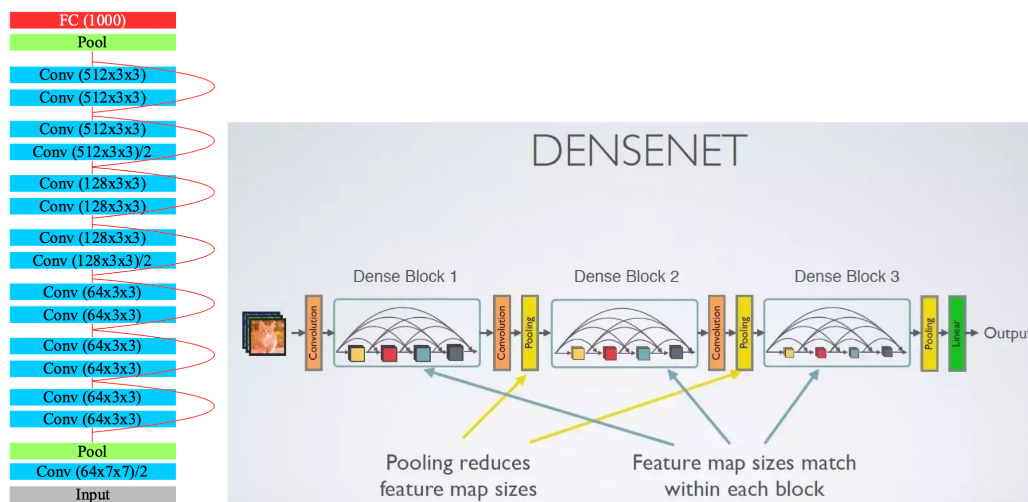
elvégzése rendkívül drága, amennyiben az aktivációs térképek csatornáinak száma nagy. Éppen ezért ezekben a hálókban minden kétdimenziós konvolúciót megelőz egy 1x1-es konvolúciós réteg, amelyik a bemeneti aktivációs térkép csatornáinak számát annak töredékére csökkenti, vagyis tömöríti. Ezt követően a kétdimenziós (3x3, vagy 5x5) konvolúciót ezen a tömörített térképen végezzük el, ezt követően egy újabb 1x1-es konvolúciós réteg ez eredeti csatornaszámot visszaállítja. Ezzel a módszerrel mind az egyes rétegek paramétereinek száma, mind a réteg végrehajtásának ideje nagymértékben csökkenthető.



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

3.3.5. DenseNet

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



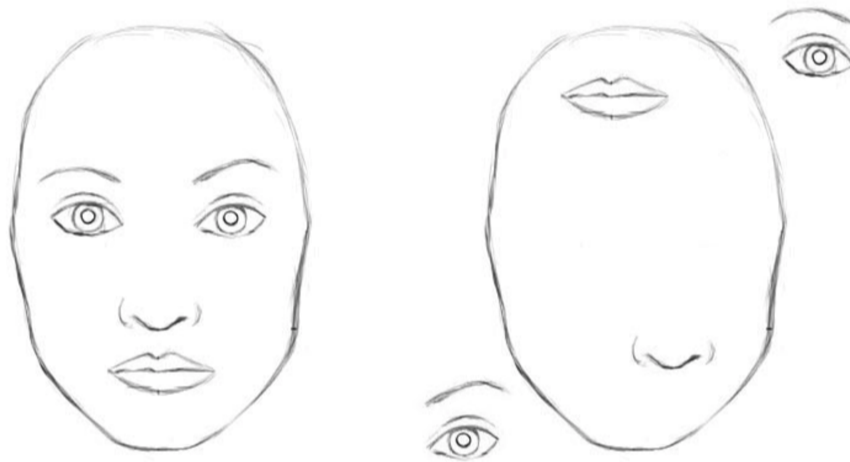
3.16. ábra. *A ResNet (bal) és a DenseNet (jobb) modellek.*

3.4. CapsNet

A konvolúciós neurális hálók a deep learning alapú képfeldolgozás leginkább elterjedt módszerei. Érdemes azonban tudni, hogy a konvolúciós architektúra a 90-es évek második fele óta létezik, az elmúlt évtizedben elsősorban a tanításuk gyakorlati praktikái fejlődtek.

Természetesen a széles körű használatnak köszönhetően az architektúra különböző hiányosságai, limitációi is napfényre kerültek. Ezek közül az egyik leginkább nyilvánvaló, hogy a konvolúciós hálók számára gyakran több száz, vagy ezer tanítóadat szükséges ahhoz, hogy egy új képi osztályt megtanuljanak felismerni, míg emberek esetén ez a szám legfeljebb egy tucat. További problémája a konvolúciós hálóknak, hogy a tanítóadatok többértelműségét (hibás címkék, több osztály előfordulása egy képen) nehezen tudják kezelni.

Ezen belül problémát jelent a (max) pooling széleskörű használata is. Ennek az operációnak a használata során ugyanis az adat térbeli méretének csökkentéséhez releváns információkat dobunk el, amelyek a térbeli struktúrák értelmezésének képességét csökkenteni fogják. Ez az (egyik) oka annak, hogy a konvolúciós hálók nem képesek jól értelmezni az általuk detektált jellemzők térbeli hierarchiáját, ami azt eredményezheti, hogy a felismeréshez fontos jellemzők helytelen elrendezését is pozitív mintaként ismerik fel. Ehhez hasonló módon a térbeli hierarchia mély ismeretének hiányában a konvolúciós hálók az egymáshoz közeli objektumok felismerésekor is gyakorta hibáznak.



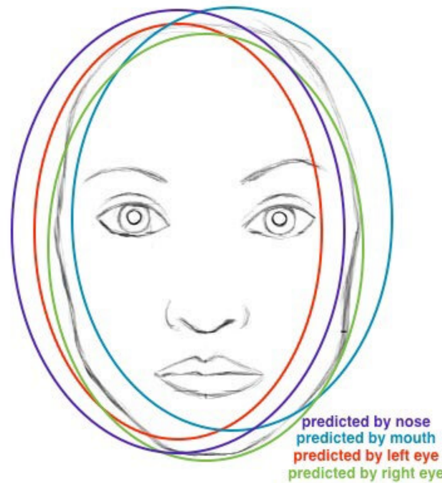
3.17. ábra. *A ConvNetek tipikus hibája: mindkét képet arcként ismerik fel.*

A konvolúciós hálózatok legfőbb problémája, hogy a térbeliség és a geometriai transzformációk nincsenek a háló struktúrába beleépítve, hanem a konvolúció egyszerűen a különböző jellemzők súlyozott átlagát veszi egy adott pozícióban (nagy kernelek képesek csekély térbeliséget vinni a műveletbe). Éppen ezért adja magát, hogy készítsük a hálókat olyan alapelemekből, amelyekben a geometria már kódolva van. Ezeket az építőelemeket kapszulának nevezzük.

A kapszula és a konvolúciós szűrőhöz hasonló módon egy képjellemző detektor egység. A kettő közti alapvető különbség, hogy a kapszula kimenete nem egy skalár, hanem egy vektor, melynek hossza a detektált jellemző erősségével arányos 0 és 1 között, iránya pedig a jellemző térbeli tulajdonságait írja le. Érdemes megjegyezni, hogy ez nem feltétlen egy háromdimenziós vektor, hiszen egy adott képjellemzőnek a pozíción és orientáción túl számos más geometriai tulajdonsága (skála, méretarányok, stb.) is lehet.

A kapszula bemenetei a háló előző szintjén lévő - alacsonyabb szintű jellemzőket detektáló - kapszulák által kiadott vektorok. A működése során a kapszula először ezeket az alacsonyabb szintű jellemzőket egy mátrixszorzás (minden bemenethez külön mátrix tartozik) segítségével transzformálja. Ez a mátrixszorzás egy geometriai transzformációval egyenértékű, ami úgy értelmezhető, hogy a kapszula megbecsüli, hogy az általa detektált jellemző hol van az alacsonyabb szintű jellemző detekció alapján.

Az ily módon előállított predikció vektorokat ezt követően a kapszula az egyes alacsonyabb szintű jellemzőkhöz tartozó súlyokkal megszorozva összeadja (ez a lépés hasonlít a konvolúciós szűrőkhöz). Ezt követően a kapszula végén egy nemlinearitás található, amely a kimeneti vektor hosszát 0-1 közé skálázza az alábbi módon:



3.18. ábra. *Az alacsonyabb szintű jellemzők (arc része) által becsült arc helyzetek.*

$$v_i = \frac{||s_i||^2}{1 + ||s_i||^2} \frac{s_i}{||s_i||} \tag{3.1}$$

Ahol v_i a kimeneti vektor értéke, s_i pedig az összegzés után előálló részeredmény. Látható, hogy a képlet jobb szélén lévő művelet a vektort egy hosszúra skálázza, míg a jobb oldal elején található tört egy további skálafaktort vezet be, amely az 0-1 közé való skálázásért felel.

Az egyetlen megmaradó kérdés már csak az, hogy hogyan határozzuk meg az alsó és felső szintű kapszulák közti súlyok értékét. Ennek megértéséhez definiálnunk kell az egyetértés fogalmát: egy alsóbb és egy felsőbb szintű kapszula között akkor van egyetértés, ha az alsó szintű kapszula által becsült felső szintű jellemző vektor és a felső kapszula kimeneti vektora hasonló irányba mutatnak. Ez ugye azt jelenti, hogy a két kapszula egyetért a detektált jellemző pozíciójában és orientációjában.

Ezt a fogalmat használja fel a dynamic routing algoritmusa, amelyet a kapszula hálók tanítására alkalmaznak. Az algoritmus két réteg között kiszámolja az egyes kapszulák aktivációit, majd a súlyok új értékét az alábbi képlettel határozza meg:

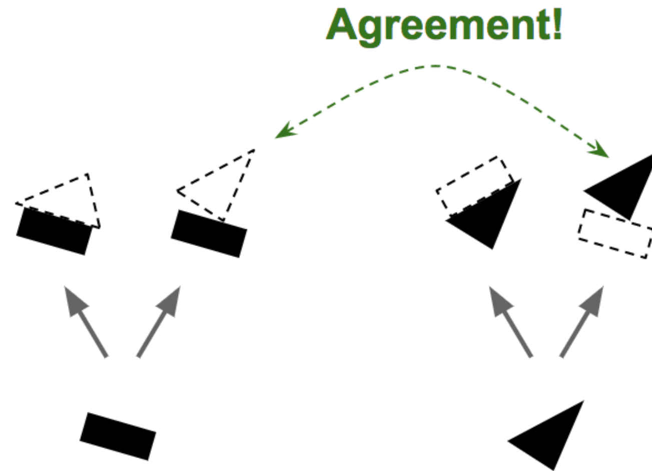
$$b_{ij} \leftarrow b_{ij} + v_i^T (W_{ij} u_j) \tag{3.2}$$

Ahol b_{ij} az i-edik és a j-edik kapszula közti súly, W_{ij} a két kapszula közti transzformációs mátrix, v_i és u_j pedig a felső és az alsó kapszulák kimenetei. Érdemes megjegyezni, hogy a két vektor közti skaláris szorzat pont a kettejük által bezárt szög koszinuszával arányos, ami tekinthető az egyetértés mértékének.

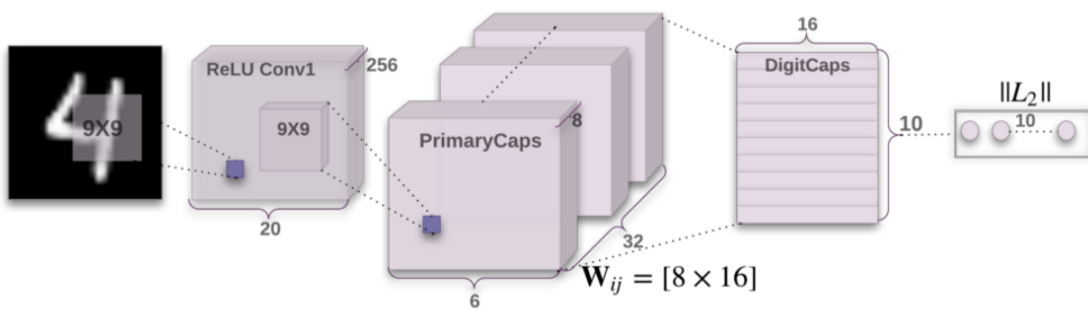
Ezek után nézzük meg, hogy miként lehetséges a kapszulák segítségével egy neurális hálót készíteni. Ehhez vizsgáljuk meg az MNIST számjegyfelismerő adatbázison használt háló architektúrát. Ez a háló egyetlen konvolúciós réteggel kezdődik, amelyet két kapszula réteg követ. Ezek közül az első egy általános rejtett réteg-szerű réteg, a második pedig a 10 számjegy detektálására késült osztályozó réteg. A hálózat kimenetén egyszerűen előírhatjuk, hogy a helyes számhoz tartozó osztályozó kimeneti vektorának hossza legyen minél nagyobb, a többi pedig legyen minél kisebb.

Ezzel azonban még csak a kapszula kimenetek méretére írtunk elő kritériumot, az irányukra azonban nem. Annak eléréséhez, hogy a kapszula kimenetek valóban tartalmazzák a detektált jellemzők/osztályok térbeli tulajdonságait, a CapsNet végére egy dekóder-hálóarchitektúrát helyezünk el, amelynek feladata, hogy a kimeneti kapszulák segítségével az eredeti képet minél pontosabbban rekonstruálni tudja.

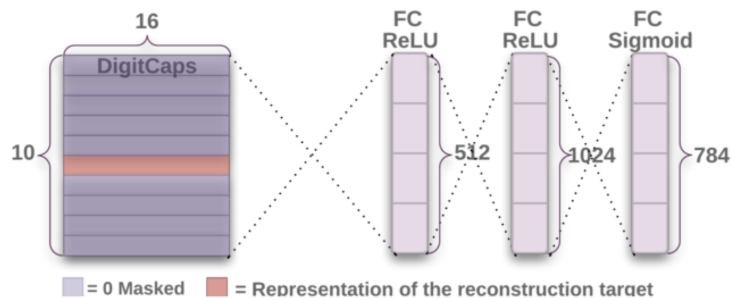
Érdemes megjegyezni, hogy a CapsNet az írott számjegyek felismerésében minden ismert konvolúciós hálónál jobb eredményt ért el. Azonban egyelőre a bonyolultabb, összetett háttérrel rendelkező



3.19. ábra. *Egyetértés esetén a súly növekszik ellenkező esetben csökken.*



3.20. ábra. *Egy tipikus CapsNet felépítése.*



3.21. ábra. *A dekóder háló-rész.*

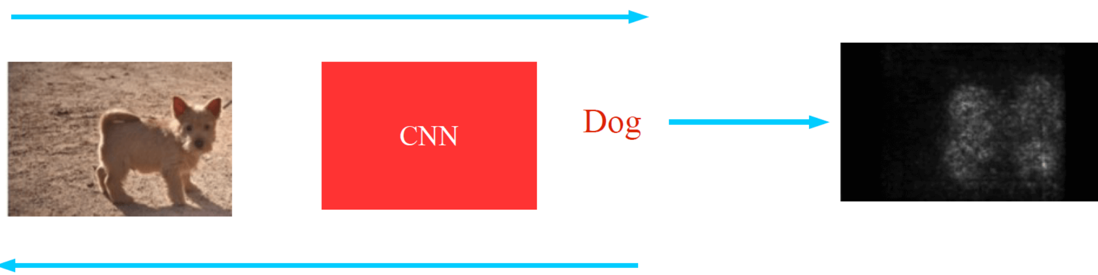
képek (pl.: CIFAR-10) problémát jelentenek az algoritmus számára. Ez azonban valószínűleg nem jelenti a kapszula hálók használhatóságának végét, elvégre a konvolúciós hálózatok esetében is jó 15 évig tartott, mire a kutatók a tanítás gyakorlati problémáit sikeresen meg tudták oldani.

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

3.5. Vizualizáció

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

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



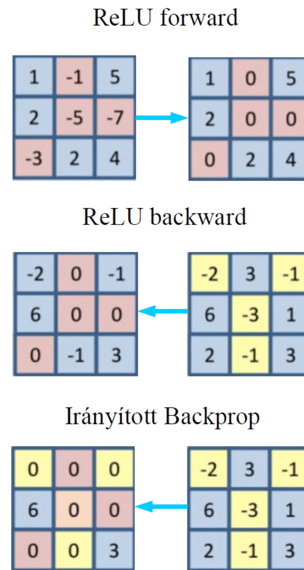
3.22. ábra. *A saliency előállítása.*

3.5.1. Guided backpropagation

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

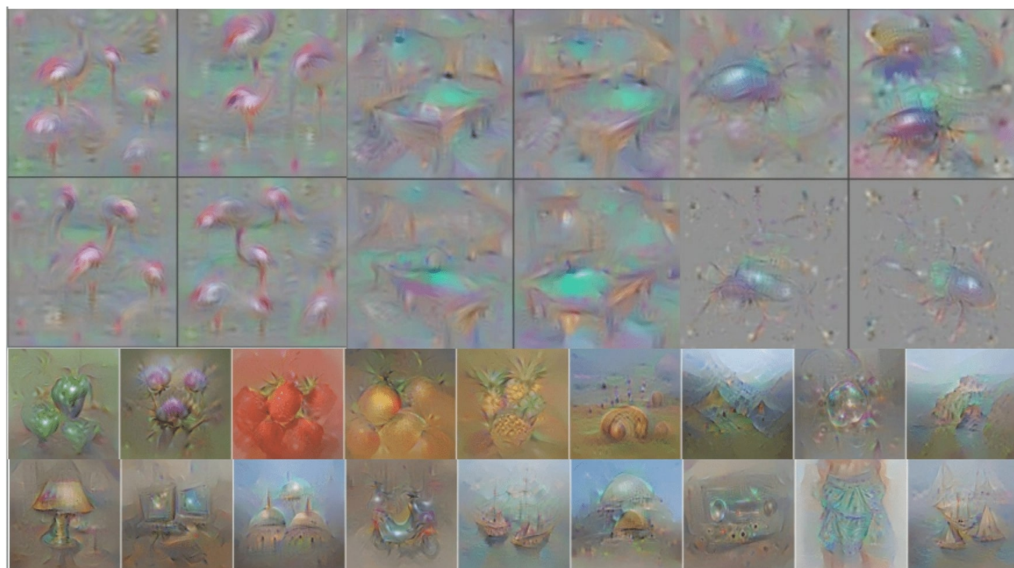
A jó minőségű saliency előállításán felül a guided backpropagation felhasználható tetszőleges neuronok vizualizációjára. Ekkor a célunk, hogy előállítsuk azt a képet, ami az adott neuront (értsd: konvolúciós szűrőt) maximálisan aktiválja. Ehhez először csupa nullával inicializáljuk a bemeneti képet,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

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



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

3.24. ábra. *A guided backpropagation elve.*

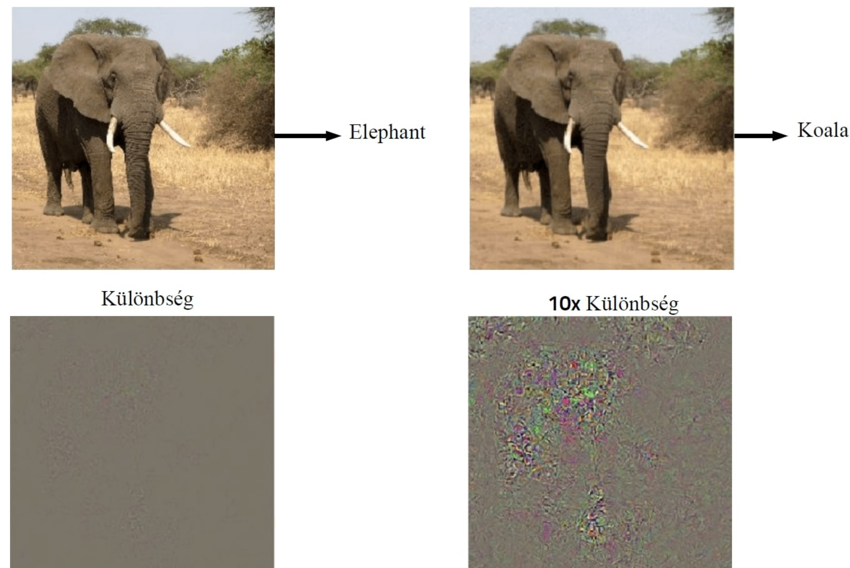


3.25. ábra. *A guided backpropagation eredménye.*

3.5.2. Ellenséges példák

A backpropagation algoritmus alkalmazásai során egy fontos felfedezés volt, hogy egy tipikus számítógépes látásban használt neurális háló esetén lehetséges helyesen osztályozott képeken olyan

minimális, emberi szem által észrevehetetlen változásokat generálni, aminek hatására a neurális háló már tévesen fogja az adott képet osztályozni. Ezeket a képeket ellenséges példának nevezzük, és jelenlegi tudásunk szerint meglehetősen nehéz ellenük védekezni. A legjobb, amit tehetünk az, hogy a tanítás során magunk generálunk ilyen példákat, és ezekkel is tanítjuk a hálót. Persze az ember sem mentes ilyen hibáktól – az emberi látás ellenséges példáit illúzióknak nevezzük. Úgy tűnik azonban, hogy a neurális hálók illúziói jelentős mértékben különböznek az emberétől.



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

3.5.3. DeepDream

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

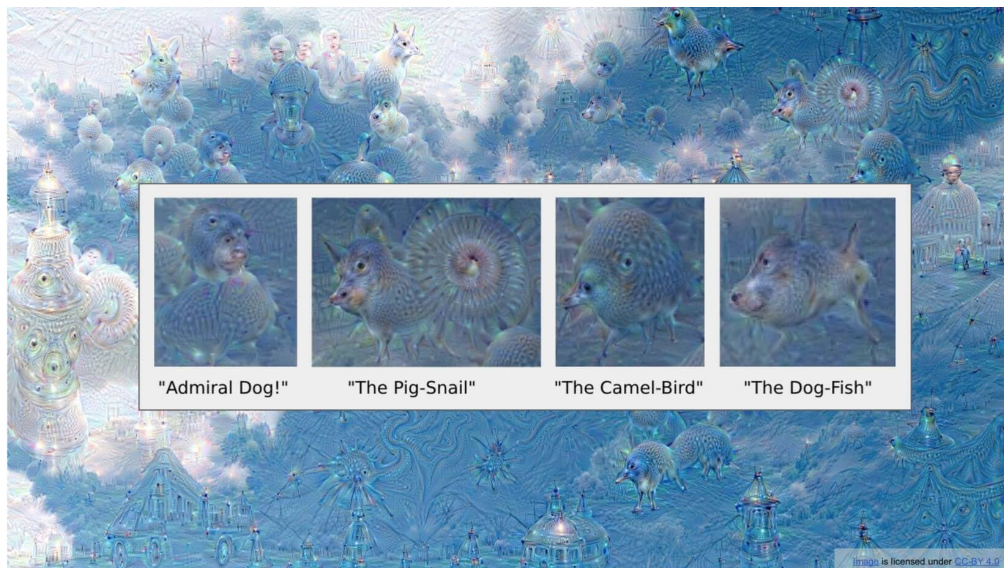
További Olvasnivaló

[1] Jeff Heaton. „Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning”. *Genetic Programming and Evolvable Machines* 19.1-2 (2017. okt.), 305–307. old. DOI: 10.1007/s10710-017-9314-z. URL: https://doi.org/10.1007%2Fs10710-017-9314-z.

[6] Christian Szegedy és tsai. „Going deeper with convolutions”. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE, 2015. jún. DOI: 10.1109/cvpr.2015.7298594. URL: https://doi.org/10.1109%2Fcvpr.2015.7298594.

[7] Richard Zhang. *Making Convolutional Networks Shift-Invariant Again.* 2019. eprint: 1904.11486. URL: http://www.arxiv.org/abs/1904.11486.

[8] Kaiming He és tsai. „Deep Residual Learning for Image Recognition”. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE, 2016. jún. DOI: 10.1109/cvpr.2016.90. URL: https://doi.org/10.1109%2Fcvpr.2016.90.



3.27. ábra. *A deep dream eredménye.*

[9] Gao Huang és tsai. „Densely Connected Convolutional Networks”. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017. júl. DOI: 10.1109/cvpr.2017.243. URL: https://doi.org/10.1109%2Fcvpr.2017.243.

[10] Sara Sabour, Nicholas Frosst és Geoffrey E Hinton. *Dynamic Routing Between Capsules*. 2017. eprint: 1710.09829. URL: http://www.arxiv.org/abs/1710.09829.

4. fejezet

Deep Learning a gyakorlatban

4.1. Bevezetés

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

A neurális hálók tanítását alapvetően négy dolog nehezíti meg:

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

4.2. Konvergencia problémák

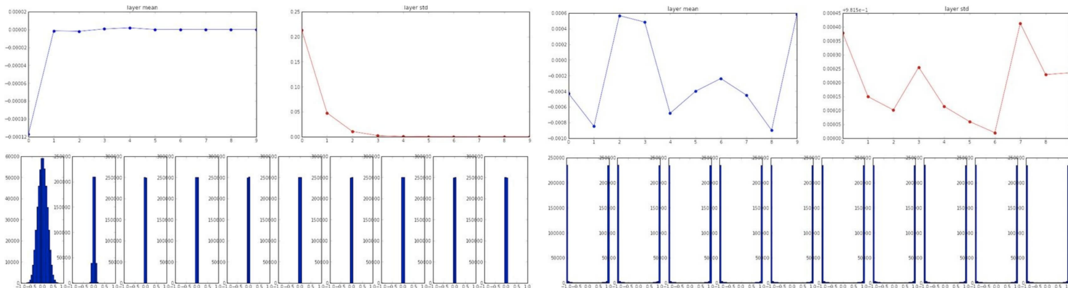
A korábbi fejezetben ismertetett gradiens módszer és backpropagation algoritmus a táblán és a tankönyvek (és ezen jegyzet) hasábjain minden esetben tökéletesen működik. A gyakorlatban azonban a lebegőpontos számábrázolás véges pontosága és tartománya miatt számos problémába ütközünk, amik az algoritmus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

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

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

4.2.1. Inicializáció

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



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

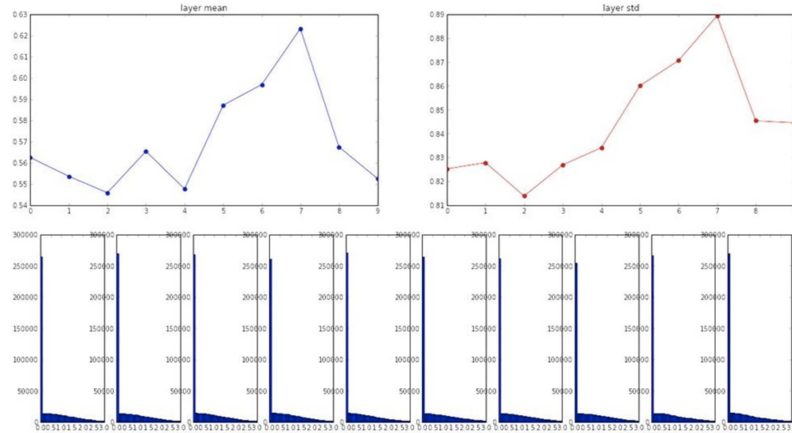
Ennek elkerülésére a háló súlyainak szórását nagy körültekintéssel kell megválasztani, hogy se túl kicsik, se túl nagyok ne legyenek. Ennek több módja is létezik, melyek közül a leginkább elterjedt választások a Xavier, illetve a He inicializációs formulák, amelyek a következőképp határozzák meg a háló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

$$\mu = 0$$
$$\sigma_{Xav} = \frac{2}{n_i + n_o} \tag{4.1}$$
$$\sigma_{He} = \frac{2}{n_i}$$

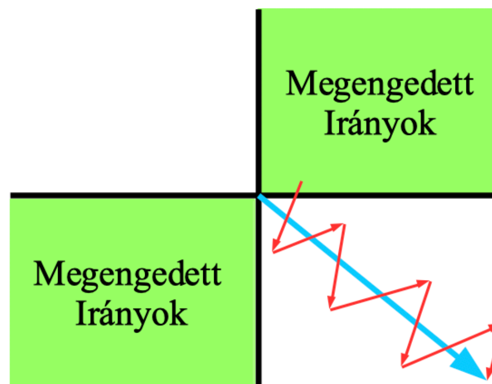
Ahol n_i és n_o az adott réteg be- és kimenteinek számát jelöli. Érdemes megjegyezni, hogy ezekkel a választásokkal a háló aktivációi és gradiensei megközelítőleg normális eloszlásúak lennének, azonban a ReLU aktivációs függvény használata ezt valamelyest torzítja.

4.2.2. Adatnormalizálás

Hasonló megfontolások miatt szükséges a neurális hálóknak bemenetként szolgáló képeken bizonyos transzformációk elvégzése. A korábbiak alapján belátható, hogy a jó numerikus konvergencia érdekében rendkívül fontos, hogy a bemenetre adott kép pixel értékeinek eloszlása megközelítőleg sztenderd normális legyen. Hiába inicializáljuk ugyanis jól a háló súlyait, ha a bemenet értékei túlságosan nagy számok, akkor hasonló problémába fogunk ütközni, mint a túl nagy súlyok esetén. Szintén fontos, hogy a pixelek átlaga a nulla közelében legyen, ugyanis csupa pozitív, vagy csupa negatív bemenet esetén a gradiens lehetséges irányait korlátozzuk.



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



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

4.3. Validáció és regularizáció

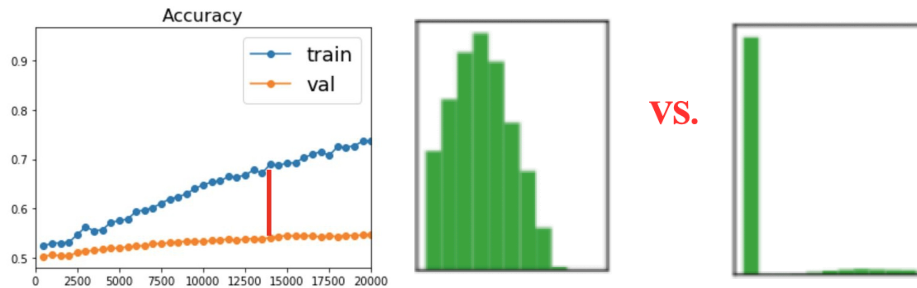
A gépi tanulás alapjainál említettük, hogy a tanítás során felléphet az overfitting jelensége. Ennek észleléséhez két külön adathalmazt érdemes használunk, egy tanító és egy validációs részt. Ezek lehetnek ugyanannak az adatbázisnak a véletlenszerűen kiválasztott részei (arányuk általában 80%-20% között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

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

4.4. ábra. *Az adatbázisok elosztása.*

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

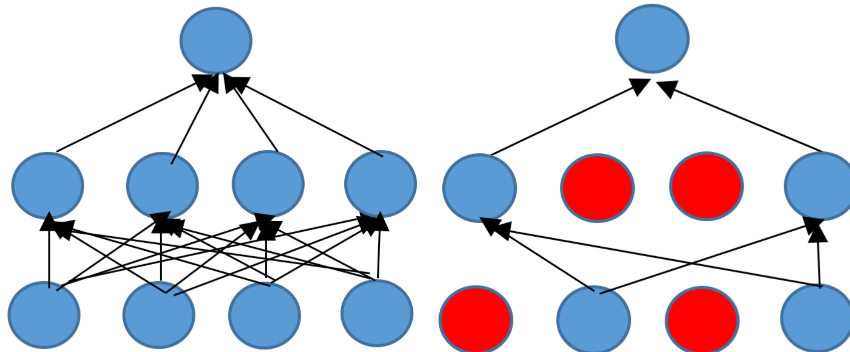
kimenetek közötti általános összefüggések helyett az egyes bemeneti tanítópéldákra adandó helyes választ kezdi el egyesével megtanulni. Ez tipikusan azt jelenti, hogy az egyes rétegekben minden egyes tanító adat esetén csak nagyon kevés aktiváció lesz maximális (amelyek épp az adott tanító példára emlékeznek),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



4.5. ábra. *Az overfitting jelenségéhez tartozó tanítási és validációs görbék (bal), valamint az aktivációk eloszlása normál (közép) és overfitting (jobb) esetén.*

4.3.1. Dropout

A nem kívánt aktivációk elkerülésére két további gyakran alkalmazott módszer létezik. Ezek közül az első a dropout nevű eljárás, melynek lényege, hogy a tanítás során minden egyes előreterjesztés során az egyes rétegek aktivációinak bizonyos hányadát véletlenszerűen kinullázzuk, és a további rétegek aktivációit így számoljuk tovább (6.6 ábra). Könnyen belátható, hogy ez a módszer meglehetősen csökkenti a túlillesztés mértékét, hiszen a hálót ezzel a módszerrel redundanciára kényszeríti. Fontos megjegyezni, hogy a tesztelés során a véletlenszerű törléseket már nem végezzük el, így viszont az egyes aktivációkat a dropout valószínűségének arányában skálázni kell.



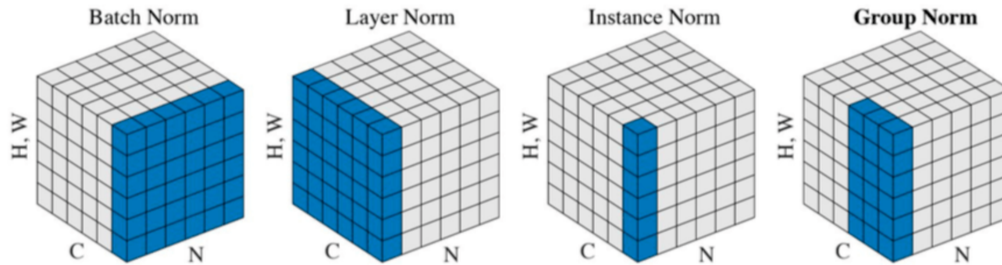
4.6. ábra. *A dropout alkalmazása.*

4.3.2. Batch Normalization

A másik megoldás az úgynevezett batch normalizálás, aminek lényege, hogy az egyes rétegek után egy normalizáló műveletet végzünk el. Korábban említettük, hogy a tanítás során egyszerre nem egy képet, hanem egy úgynevezett minibatch-nek megfelelő (általában 32 többszöröse) képet értékelünk ki. A batch normalizálás műveletének lényege, hogy az egyes aktivációk átlagát és szórását a tanítás során folyamatosan számoljuk, és az egyes aktivációkat ennek segítségével normalizáljuk. Érdemes megjegyezni, hogy ez a művelet nem csak a túlillesztést mérsékeli az aktivációk eloszlásának normalizálásával, hanem a numerikus konvergenciát is javítja. A batch normalizálás képlete az alábbi:

$$x_{BN} = \frac{x - \mu}{\sigma^2 + \epsilon} \quad \leftarrow \quad vanilla$$
$$x_{AF} = \alpha x_{BN} + \beta \quad \leftarrow \quad affin \tag{4.2}$$
$$x_{DC} = \Sigma^{-\frac{1}{2}}(x - \mu) \quad \leftarrow \quad decorrelated$$

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



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

A Batch Normalization működésének miértje nem ismert pontosan, azonban elég jó sejtéseink azért vannak. Az overfitting elkerülésében alapvetően azért segíthet, mert a tanítás során az egy batchbe tartozó képek véletlenszerűen kerülnek kiválasztásra. Ez azt jelenti, hogy egy adott kép minden alkalommal más képekkel együtt kerül a háló bemenetére, így az egyes rétegek aktivációi során az átlagok és a szórások is mások lesznek. Ennek következtében minden egyes epochban ugyanazt a képet más tényezőkkel normalizáljuk, amely nehezíti a magolást.

A konvergencia javítása kicsit nehezebb kérdés, itt ugyanis két hatás társul. Egyrészt, a normalizálás stabilizálja a neurális hálót, a forward művelet során nem lesznek túl kicsi, vagy túl nagy aktivációk. Másrészt, a Batch Normalization segít elkerülni az ún. Internal Convariate Shift (ICS) jelenségét. Az ICS abból fakad, hogy a neurális háló összes rétegét egyszerre tanítjuk. Így minden egyes tanítási lépésnél az adott réteg megpróbál jobb válaszokat adni az előző réteg által szolgáltatott bemenetekre. A gradiens lépés után azonban az előző réteg is megváltozik, és már más bemeneteket fog produkálni, melynek eredményeképp a következő rétegnek ahhoz megint adaptálódni kellene. A Batch Normalization azonban fix eloszlást erőltet rá az egyes rétegek kimenetére, így a rétegeknek sokkal kevesebbet kell "egymás után szaladgálniuk".

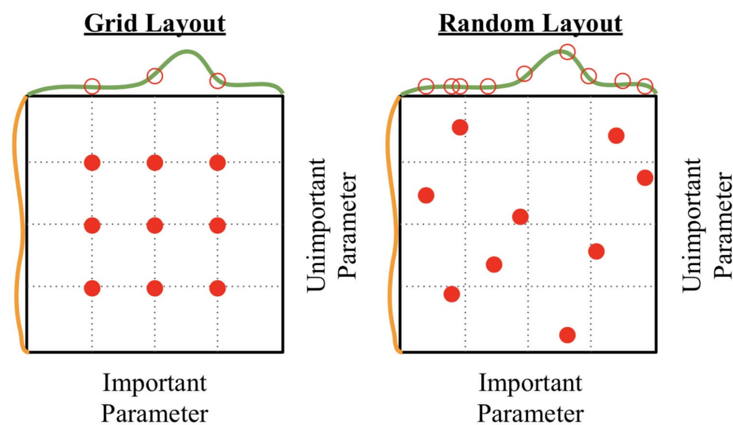
4.3.3. Adat Augmentáció

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

4.4. Hiperparaméter optimalizáció

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

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



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

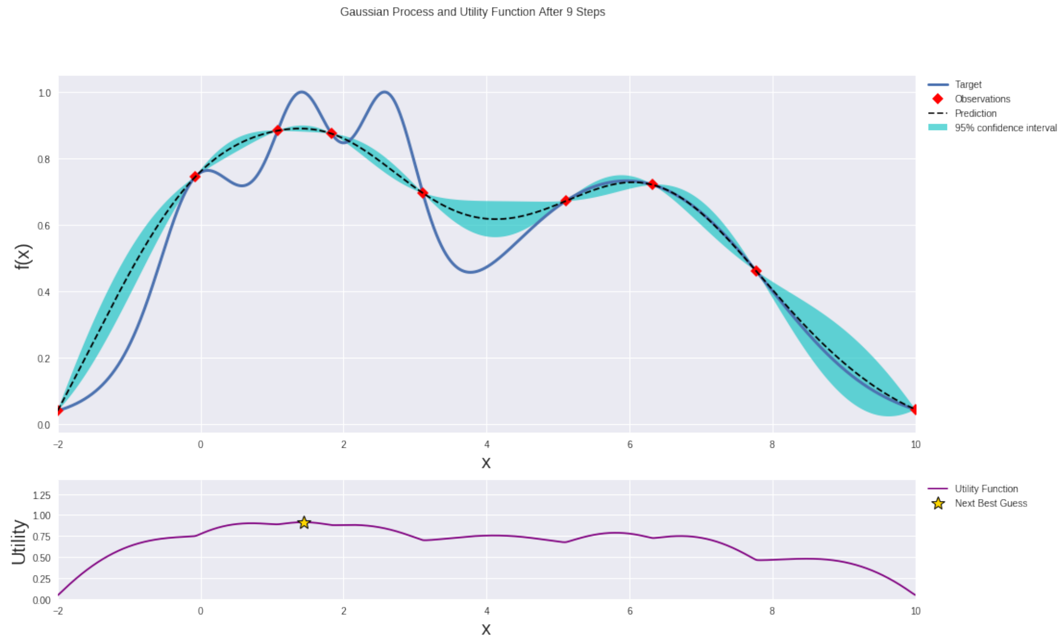
4.4.1. Bayesi optimalizáció

Habár hiperparaméter optimalizálást általában csak valamilyen véletlen próblálkozásra épülő sémával lehet végezni, ezt azonban (ahogy azt az előbbiekben is láthattuk) egyáltalán nem mindegy, hogy hogyan végezzük el. Egy igazán jó stratégia ugyanis sokkal kevesebb próblálkozásból képes lehet egy jó hiperparaméter kombinációt megtalálni. Az egyik leghatékonyabb ilyen stratégia az ún. Bayesi optimalizáció. Ennek lényege, hogy a problémára úgy tekintünk, mint egy skalárértékű függvény maximumkeresésére: a függvény bemenetei a hiperparaméterek, kimenete pedig az adott hiperparaméterekkel elérhető validációs pontosság.

Az algoritmus kezdetben végez néhány inicializációs lépést, véletlen választott paraméterekkel, majd ezek alapján becslést ad a függvény alakjára. A becslésnek minden pontban van egy várható értéke és egy szórása/konfidenciája. Innentől kezdve az optimalizálás célja kettős: Egyrészt, szeretnénk olyan új pontokat kipróbálni, ami az aktuális becslésünk alapján valószínűleg jó, de mivel a becslést tudjuk minden lépés után finomítani az új adattal, ezért érdemes a nagy bizonytalanságú pontokat is bejárni, hiszen az itt végzett mérésekkel tanulhatunk a legtöbbet a függvényről. Ennek a két kritériumnak a súlyozott kombinációját hívjuk hasznosságnak (utility).

A Bayesi optimalizálás lépései tehát a következők:

1. n Kezdeti mérés elvégzése random pontokban.

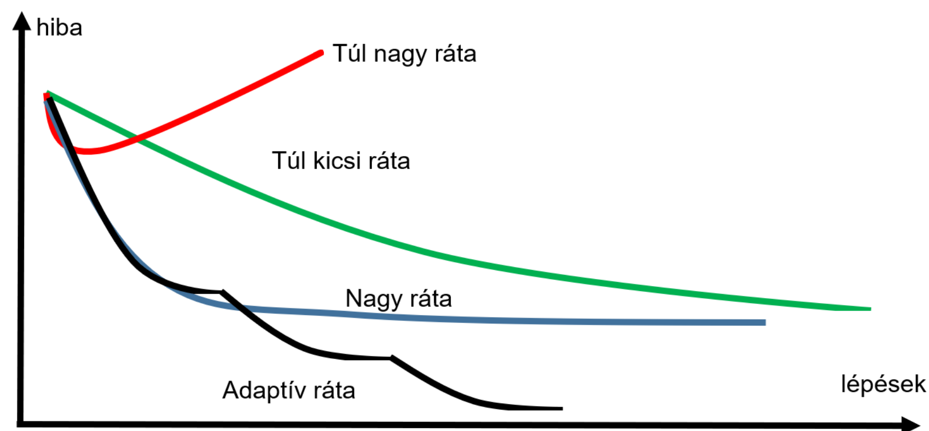


4.9. ábra. *A Bayesi optimalizációs során becsült függvény (felül) és a hasznosság (alul).*

2. Ismételjük N lépésig:
   (a) A függvény becslése, konfidencia számolása a meglévő mérések alapján.
   (b) A legnagyobb hasznosságú pont kiértékelése.

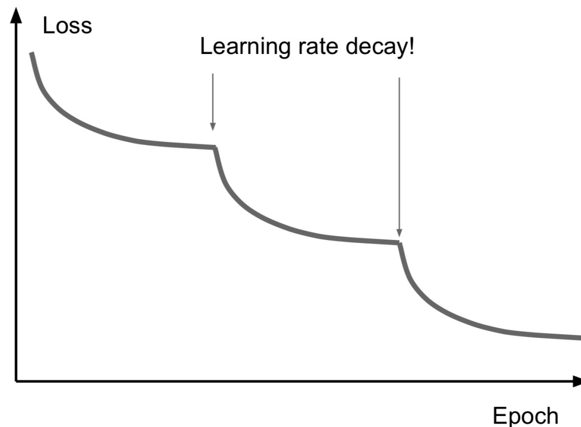
4.4.2. Tanulási ráta

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



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

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



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

Lehetőség van természetesen a ráta adaptív állítására a tanulási sebesség folyamatos monitorozása által. Ekkor a rátát akkor csökkentjük egy fix faktorral, amennyiben a tanulás már bizonyos számú lépés óta nem tudta a korábbi legjobb eredményét meghaladni. Szintén hatásos módszer a koszinuszos lágyítás alkalmazása, ami a tanulási rátát egy maximum és minimum érték között egy koszinusz függvény első fél periódusának megfelelően állítja. A koszinusz lágyítás alkalmazásakor a félperiódus befejezésekor tovább lehet folytatni a tanítást a maximális tanulási értéket használva és újabb lágyítást végezni. Ennek értelme, hogy a hirtelen megnövelt tanulási ráta „kilöki” a háló súlyait a lokális minimumból és a további tanulás során egy közeli jobb lokális minimum megtalálását teszi lehetővé.

4.5. Adatbázisok előállítása

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

4.5.1. Transfer learning

A mély tanulás területének egyik legjelentősebb áttörése ezt a problémát orvosolja. Beláttuk ugyanis, hogy a konvolúciós neurális hálók első konvolúciós és leskálázó rétegekből álló része különböző képi jellemzők detektálását végzi el. Ahogy előrefele haladunk a háló rétegei között úgy ezek a jellemzők egyre komplexebbé, egyre feladatspecifikusabbá válnak. Ebből következik azonban, hogy ha már van egy valamilyen feladatra betanított hálózatunk, akkor annak elülső rétegei felhasználhatók egy másik, hasonló feladat elvégzésére. Így elegendő az új feladathoz csak a háló utolsó rétegeit újra tanítani, amihez lényegesen kevesebb adat elegendő, hiszen kevesebb szabad paramétert tartalmaznak, mint az egész háló.

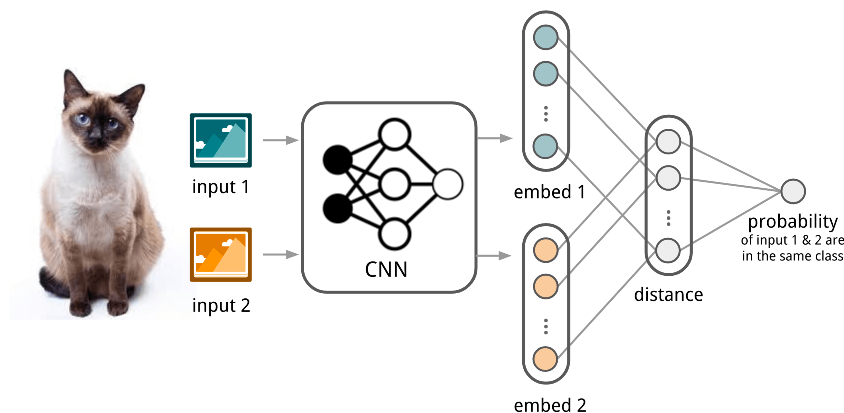
Ezt a technikát transzfer tanulásnak nevezzük, és elterjedt megoldás a mély tanulás területén. A fent ismertetett érvelés annyira igaz, hogy számos esetben elegendő a hálók legutolsó, lineáris

rétegét újra tanítani. Más esetekben szükség lehet az elülső hálórészek finomhangolására, azonban ehhez is nagyságrendekkel kevesebb adat elegendő lehet. Egy további technika a hálók teljesítményének növelésére a modell együttesek használata. Ekkor több, függetlenül tanított (és gyakran eltérő architektúrájú) neurális háló eredményeit átlagoljuk össze, ami a pontosságot akár 2-3%-kal is növelheti. Az együttesek használata a korábban említett ellenséges példák ellen is nyújthat valamekkora védelmet.

4.5.2. Meta learning

A nagy tanító adatbázis igényét az úgynevezett meta tanulás módszereivel is csökkenthetjük. A meta tanulás definíciója jelenleg nem kiforrott, de alapvetően arról van szó, hogy a tanuló algoritmus nem egy problémát megoldani, hanem általánosan tanulni tanul meg. Ebbe a meglehetősen lazán definiált problémakörbe több lehetséges megoldáscsalád is tartozik, melyek közül az első az úgynevezett metrikus tanulás. A metrikus tanulás célja, hogy megtanuljuk a képeket egy olyan alacsonydimenziós jellemző vektorral leírni, amelyek kompakt módon tartalmazzák a kép összes fontos tulajdonságát. Ha a háló egy ilyen reprezentációt képes megtanulni, akkor általános használható majd képek felismerésére.

Metrikus taulás egyik példája az úgynevezett sziámi-háló. Ennek a hálónak a bemenetére két képet adunk, és mindkét képből külön-külön előállít a kimenetén egy leíró vektort. Ezt követően a hálót arra tanítjuk, hogy az azonos osztályba tartozó képek leíró vektorai között legyen minél kisebb a távolság, az ellenkező osztályokba tartozók pedig legyenek minél messzebb egymástól.



4.12. ábra. *A sziámi hálók struktúrája.*

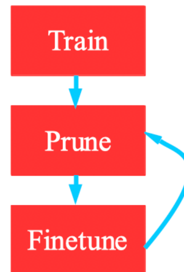
A metrikus tanuláson felül léteznek olyan neurális hálózatok amelyeket arra tanítják, hogy képesek legyenek az emberi tervezőkhöz hasonló módon jó modellstruktúrát és hiperparaméter értékeket találni egy adott tanítási feladathoz a lehető legkevesebb próbálkozás alapján. Ezek hálók általában visszacsatolt struktúrájúak, vagyis képesek változó hosszúságú szekvenciákat generálni, ezáltal egy egész neurális háló struktúrát képesek előállítani a kimenetükön. Ezeket a hálókat általában a később tárgyalt megerősítéses tanulás segítségével tanítják.

4.6. Installáció

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

4.6.1. Pruning

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

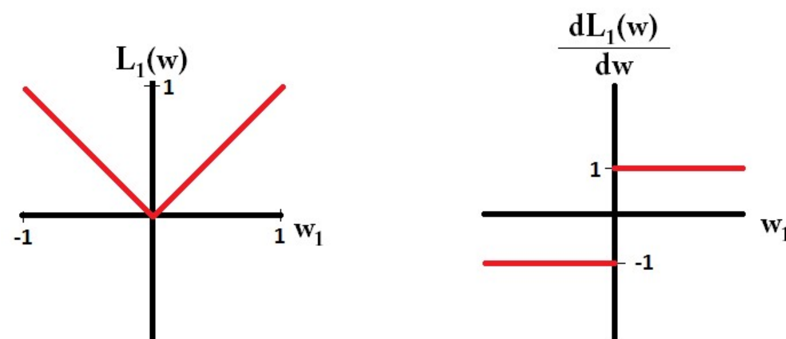


4.13. ábra. *A Pruning módszere.*

Lasso regularizáció

Érdemes megjegyezni, hogy a pruninghoz hasonló hatást a regularizáció segítségével is elérhetünk. Ehhez azonban nem mindegy, hogy milyen típusú regularizációt használunk. Mint tudjuk a regularizáció célja, hogy a súlymátrix normája minél kisebb maradjon, vagyis a regularizációs tag hatására a súlyok a nullához közelebb kerülnek. Ezt a regularizáció úgy éri el, hogy a költségfüggvényhez egy büntetőtagot ad hozzá, így a gradiens módszer során ennek a gradiense is levonásra kerül a súlyokból.

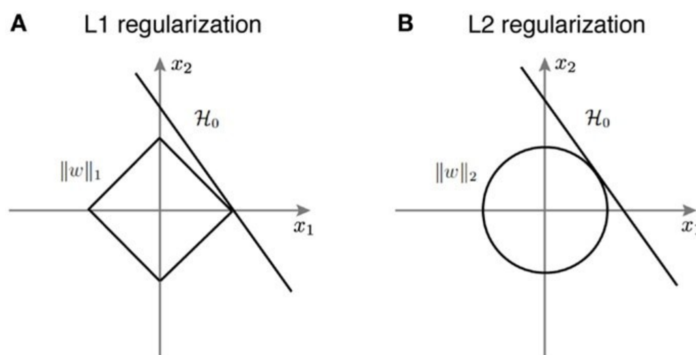
Észrevehető azonban, hogy a különböző regularizációs formáknak azonban - nyilvánvalóan - eltérő a deriváltja. Érdemes ezt a gyakran használt L1 és L2 regularizációk példáján szemléltetni. Az L2 regularizáció a súlyok négyzetével arányos, így a deriváltja magával a súllyal arányos. Ennek az a következménye, hogy minél közelebb kerül egy súly a nullához, annál kisebb hatást fejt ki rá a regularizáció. Az L1 regularizáció azonban magával a súllyal arányos, így a deriváltja egy konstans, melynek előjele a súly előjelétől függ. Ez azt jelenti, hogy az L1 regularizáció minden súlyra annak nagyságától függetlenül hat, így a súlyok jóval nagyobb részét fogja kinullázni (vagy legalábbis hibahatáron belüli közelségbe vinni). Éppen ezért az L1 reguralizációt gyakorta illesztjük a ritkító jelzővel.



4.14. ábra. *Az L2 (bal) és az L1 (jobb) normák deriváltjai.*

Érdemes megjegyezni, hogy a konvolúciós hálókban gyakorta nem az egyes súlyokat regularizáljuk, hanem az egyes konvolúciós mátrixok normáit összeadva a teljes mátrixokkal tesszük meg ugyanezt.

Ennek az a következménye, hogy az egyes súlyok helyett teljes konvolúciós szűrőket nullázunk ki, így pedig sokkal hatékonyabban tudjuk a hálót optimalizálni, különösképp párhuzamos hardver architektúrák esetében. Az L1 regularizációt gyakran nevezzük LASSO (Least Absolute Shrinkage and Selection Operator) regularizációnak.



4.15. ábra. *Az L1 regularizáció ritkító hatása: Mindkét regularizáció esetében a súlyok megválasztására számos (a költségfüggvény szempontjából) egyenlően jó megoldásunk adódik, melyek ebben a példában az ábrán látható egyenes mentén helyezkednek el. Bal oldali ábrán középen található négyzetes görbe az L' kritérium alapján egyenlő jó megoldásokat ábrázolja, míg a jobb oldali kör az L" kritérium alapján egyenlő megoldásokat. Látható, hogy hacsak az egyenlő jó megoldásokat tartalmazó egyenes nem pontosan 45 fokos, akkor a regularizáció során az egyik változó nullázódni fog.*

4.6.2. Weight sharing

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

A weight sharing egy egyszerűbb változata a közönséges kvantálás, ahol a súlyokat egyszerűen kevesebb bit felhasználásával alkalmazzuk. Ennek gyakori változata a 16 bites lebegőpontos (half), valamint a 8, 4, 2, illetve 1 bites fixpontos számok használata. Ennek külön előnye az, hogy a háló súlyinak tömörítésén felül gyorsításra is lehetőséget ad, hiszen a számolást is kevesebb biten kell elvégezni. A legtöbb modern processzor támogatja az úgynevezett SWAR (SIMD Within a Register) megoldásokat, melyek lényege, hogy egyetlen 64 bites ALU segítségével egy ciklus alatt több alacsonyabb bitszámú művelet elvégezhető.

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

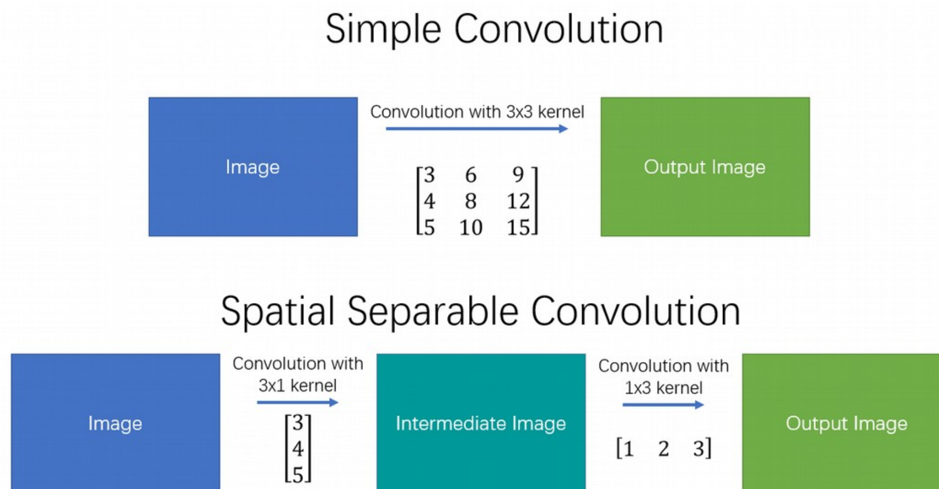
4.6.3. Ensemble

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

4.6.4. Szeparálható konvolúció

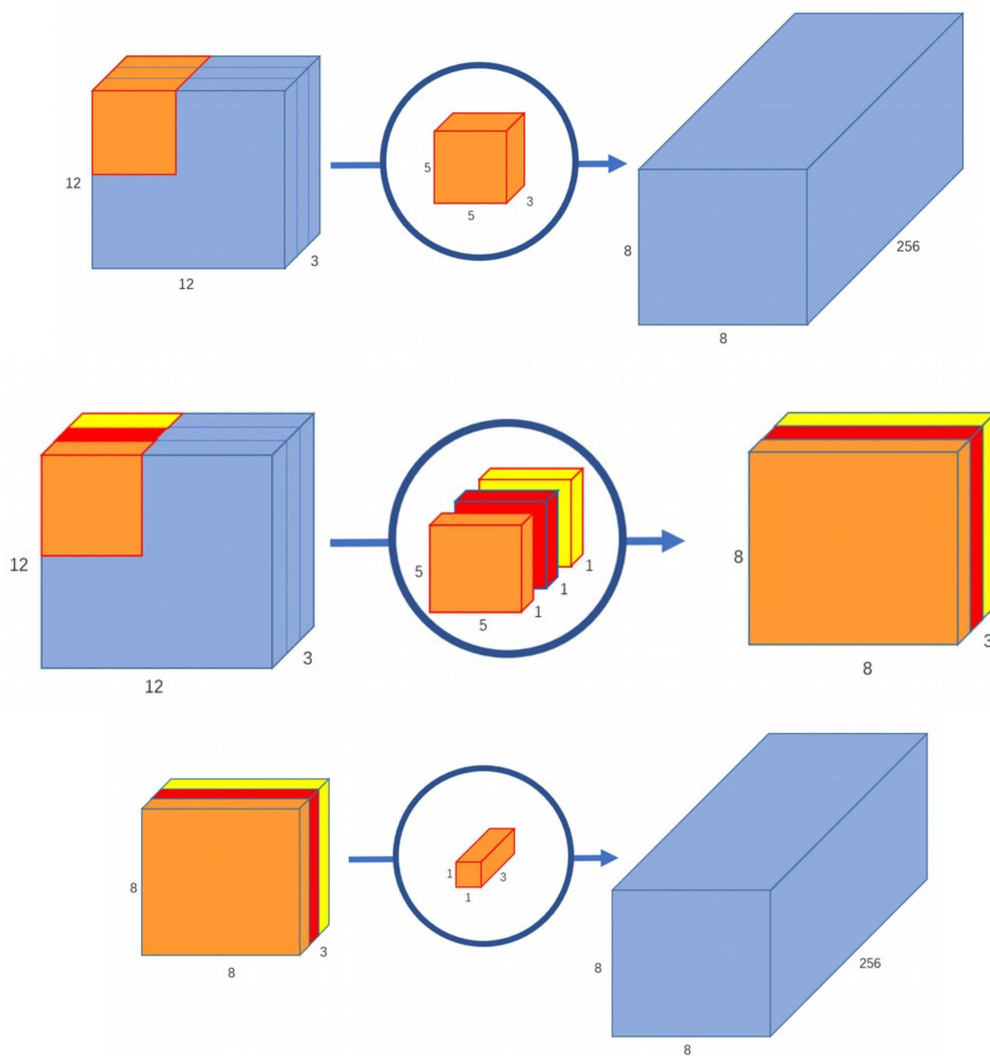
Fontos még megemlíteni a neurális hálók gyorsításának egy utolsó lehetőségét, amely a háló rétegeinek súlymátrixainak alacsony rangú faktorizációjára alapul (low-rank matrix factorization). Ennek az eljárásnak lényege, hogy a súlymátrixokat diadikus szorzatok összegére bontjuk. Triviálisan minden $N \times M$ mátrix felbontható $min(N, M)$ darab diadikus szorzat összegére, azonban ha az adott mátrix rangja (r) ennél kisebb, akkor ennyi is elég. Ennek a felbontásnak a meghatározására gyakorta használják a mátrixok úgynevezett szinguláris érték felbontását (SVD), amely minden egyes diádhoz egy ”erősség” értéket rendel (ezt nevezzük szinguláris értéknek), amely megadja, hogy mekkora lenne a hiba, ha az adott diádot elhagynánk. Ezek segítéségével a diádok száma tovább csökkenthető úgy, hogy az ez által okozott hiba minimális legyen.

A diadikus felbontást konvolúciós szűrők esetében több dimenzió mentén is felhasználhatjuk. A konvolúciós szűrések esetében ugyanis valójában nem egy egyszerű súlymátrixunk van, hanem egy 4 dimenziós súly tenzorunk ($k_W \times k_H \times ch_{in} \times ch_{out}$), így a felbontást több dimenzió mentén is elvégezhetjük. Ennek az egyszerűbb változata a térbeli szeparáció, amikor a 2D konvolúciót két darab egy dimenziós konvolúció szorzatára bontjuk, így csökkentve a számítási kapacitást.



4.17. ábra. *A térben szeparált konvolúció elve.*

Ennek egy másik fajtája az úgynevezett mélység szerint szeparált (depth-wise separable) konvolúció. Ebben az esetben először a bemeneti aktivációs tömb egyes csatornáin külön-külön végzünk el egy 1 csatornás konvolúciót. Az így keletkezett aktivációs tömbön ezt követően egy 1×1 méretű, de már az összes csatornát egyszerre használó konvolúciót futtatunk. Miért is éri ez meg? Számoljuk ki a konvolúciós réteg futtatásához szükséges MaC (Multiply and Accumulate) műveletek számát: A normál esetben ez $k_w * k_h * ch_{in} * ch_{out}$, a mélység szerint szeparált esetben viszont $k_w * k_h * ch_{in} + ch_{in} * ch_{out}$. Látható, hogy ez rendkívül előnyös, különösképp, ha a kernel mérete és/vagy a kimeneti csatornák száma nagy.



4.18. ábra. *A mélység szerint szeparált konvolúció elve. Hagyományos konvolúció (felül), a szeparált változat két lépése (középen és alul).*

További Olvasnivaló

[1] Jeff Heaton. „Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning". *Genetic Programming and Evolvable Machines* 19.1-2 (2017. okt.), 305–307. old. DOI: 10.1007/s10710-017-9314-z. URL: https://doi.org/10.1007%2Fs10710-017-9314-z.

[11] Kaiming He és tsai. „Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". *2015 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2015. dec. DOI: 10.1109/iccv.2015.123. URL: https://doi.org/10.1109%2Ficcv.2015.123.

[12] Sergey Ioffe és Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. eprint: 1502.03167. URL: http://www.arxiv.org/abs/1502.03167.

[13] Wei Wen és tsai. *Learning Structured Sparsity in Deep Neural Networks*. 2016. eprint: 1608.03665. URL: http://www.arxiv.org/abs/1608.03665.

II. rész

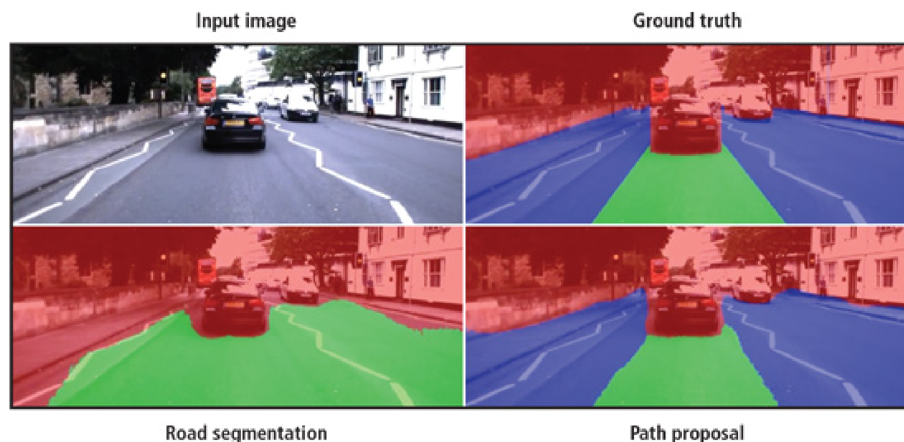
Magasszintű Látási Feladatok

5. fejezet

RNN és Transzformer

5.1. Magasszintű látási feladatok

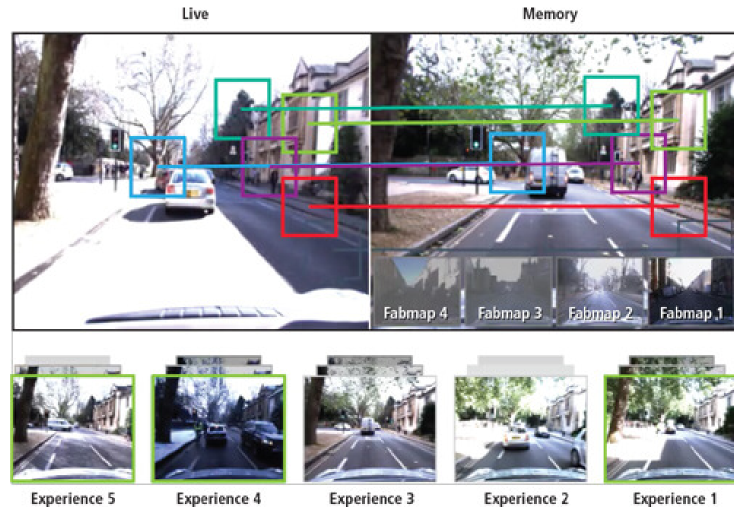
Az korábbi előadásokban kizárólag az osztályozás problémájának megoldását tárgyaltuk, mivel ez a magasszintű számítógépes látás legegyszerűbb alapproblémája. Könnyen belátható azonban, hogy az osztályozás felhasználhatósága meglehetősen limitált, két okból is. Egyrészt a képből kinyert információ meglehetősen kevés: pusztán egy adott kategória jelenlétéből ritkán lehetséges fontos döntéseket hozni. Gondoljunk itt az önvezető járművek problémájára: ebben az esetben létfontosságú a különböző kategóriák (úttest, jármű, gyalogos, tábla) pontos pozíciójáról, és méretéről is információt szerezni. Éppen ezért ilyen esetekben nem az osztályozás, hanem a detektálás vagy a (szemantikus/példány) szegmentálás feladatát kell megoldanunk.



5.1. ábra. *A szegmentálás felhasználása útvonaltervezésre.*

Az osztályozás, detektálás és szegmentálás feladatainak van azonban egy másik problémája is: Ezek ugyanis az egymás után kapott képkockákat teljesen független bemenetekként kezelik, és mivel a hagyományos konvolúciós neurális hálóknak nincs memóriaelemük, így nem is képesek emlékezni a korábbi bemenetekre. Könnyen belátható azonban, hogy ez a jelen esetben nem elég: Járművezetés esetén a neurális háló valójában egy összefüggő képsorozatot kap a bemenetére, melyek közti összefüggések felhasználhatók lennének a döntéshozáshoz.

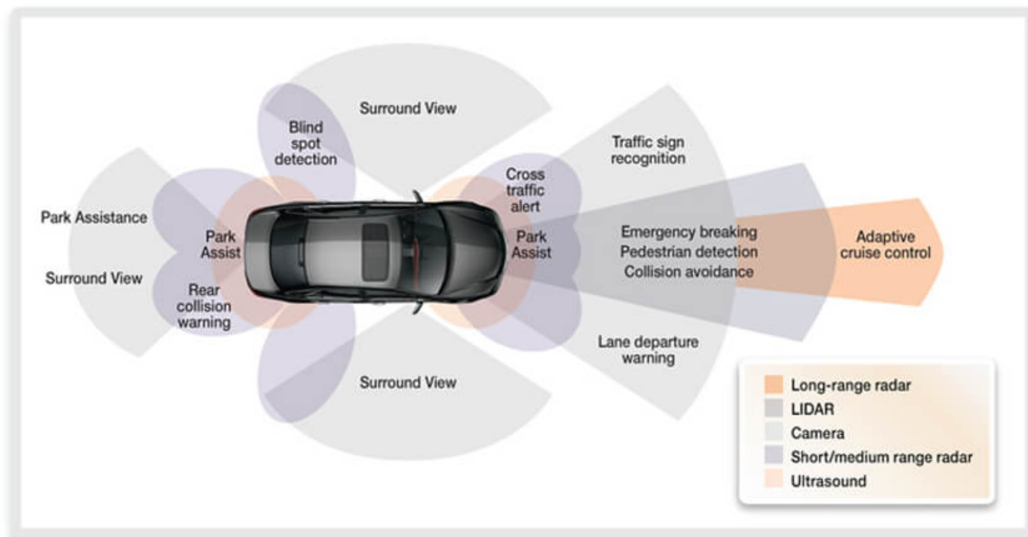
A felhasználási lehetőségekre a legegyszerűbb példa a különböző tárgyak mozgásának követése és predikciója. Ez az információ felhasználható akár arra, hogy a közlekedési szituáció többi szereplőjének a szándékát meg tudjuk becsülni, és ez által a cselekedeteikre előre felkészülve nagy mértékben csökkentsük a jármű reakcióidejét. Elképzelhető még olyan alkalmazás is, ahol a jármű a már meglátogatott helyekről emlékeket alkot, majd - az emberhez hasonlóan - ezen emlékek segítségével offline is képes magát lokalizálni.



5.2. ábra. *Párosítás az aktuális kamerakép és az emlékként eltárolt kép között.*

Autonóm járművek esetében további nehézséget okoz, hogy általában nem egy képi (vagy ahhoz hasonló jellegű) információt szolgáltató szenzor áll rendelkezésre, hanem több. Ezek általában az autó környezetének külön-külön részleteiről szolgáltatnak információt, de előfordulnak több szenzor által lefedett térrészletek is. Ezen felül az egyes szenzorok által szolgáltatott képek alapvető tulajdonságai (felbontás, bitmélység, látószög, zoom faktor stb.), vagy akár a dimenzionalitás (2D, 3D) is különbözhetnek.

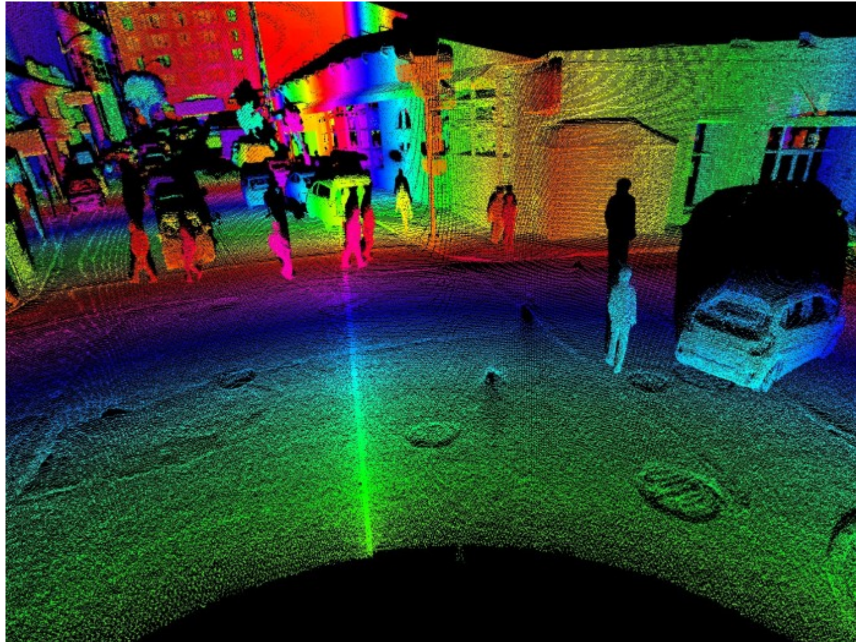
A jármű döntéshozó rendszerének fejlesztéséhez azonban a különböző szenzorokból származó információ valamilyen módon integrálnunk, fuzionálnunk kell. A szenzorok közi átfedések esetében a kinyert információt pontosíthatjuk, mivel több mérés áll rendelkezése, míg a különálló részek esetében az információk illesztésével bővíthetjük a környezetről alkotott képet, képesek lehetünk az egyes objektumokat azok teljes kontextusában értelmezni. Ezt a problémát nevezzük szenzorfúziónak.



5.3. ábra. *Egy tipikus autonóm jármű szenzorrendszere.*

A fenti ábrán látható tipikus szenzorrendszert tanulmányozva feltűnhet, hogy autonóm járművek esetén gyakori a különböző 3D szenzorok alkalmazása (ez nem meglepő, hiszen a járművezetés során az egyes objektumok távolsága létfontosságú információ). Ez azonban felvet egy újabb problémát: hogyan lehetséges neurális hálózatok segítségével 3D ”képeket” feldolgozni?

Az elkövetkező három előadás célja, hogy a korábban felvetett három problémára (detektálás-szegmentálás, videók és 3D információ) adott tipikus megoldásokat ismertesse, valamint, hogy ezek nehézségeibe, kritikus problémáiba betekintést adjon.



5.4. ábra. *Egy LIDAR által készített 3D struktúra.*

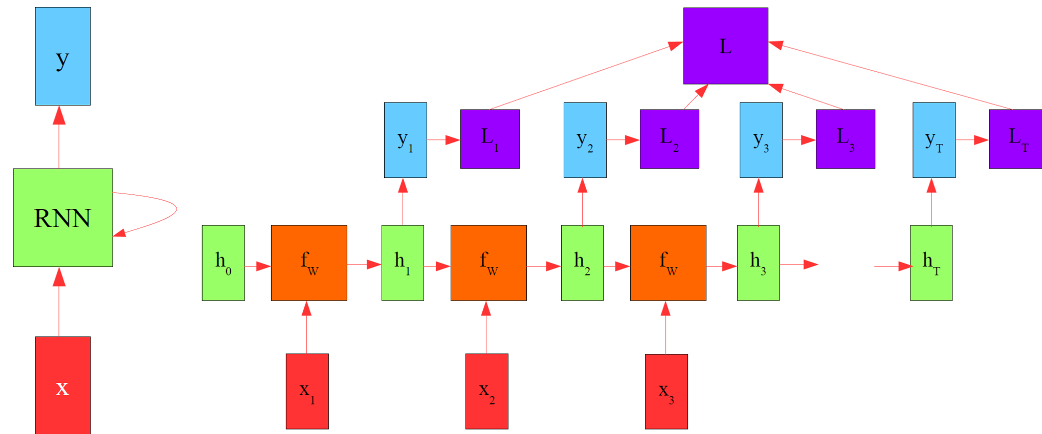
5.2. Visszacsatolt neurális hálók

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

$$\begin{aligned} h_t &= tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ y_t &= \sigma(W_{hy}h_t \end{aligned} \tag{5.1}$$

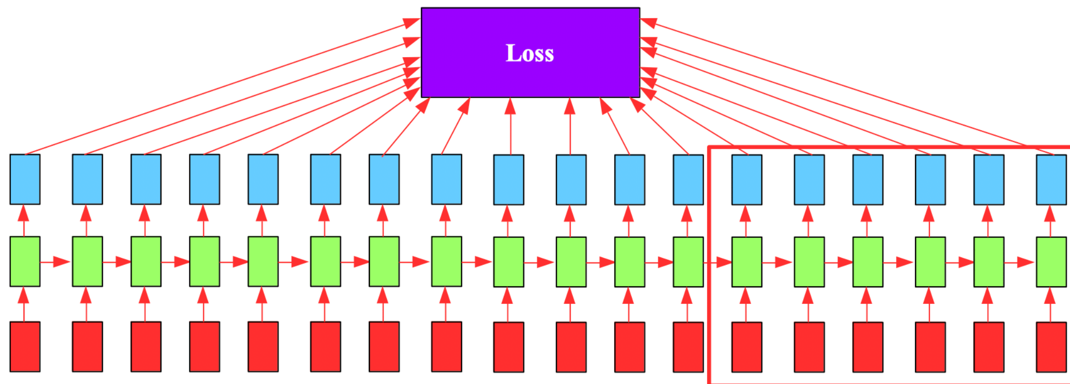
Ahol h a belső állapotot jelöli, t pedig az aktuális időpillanat. Könnyen belátható, hogy egy RNN cella tulajdonképpen három lineáris réteg és egy aktivációs függvény együtteseként adódik. Az új architektúra bevezetésével azonban felmerül az a kérdés, hogy hogyan lehet ebben a struktúrában a súlyok gradienseit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

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



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

bemenetekre. Éppen ezért a kibontás során a háló maximális méretét korlátozzuk és a legrégebbi réteget és bemenetet töröljük a kibontott hálóból.



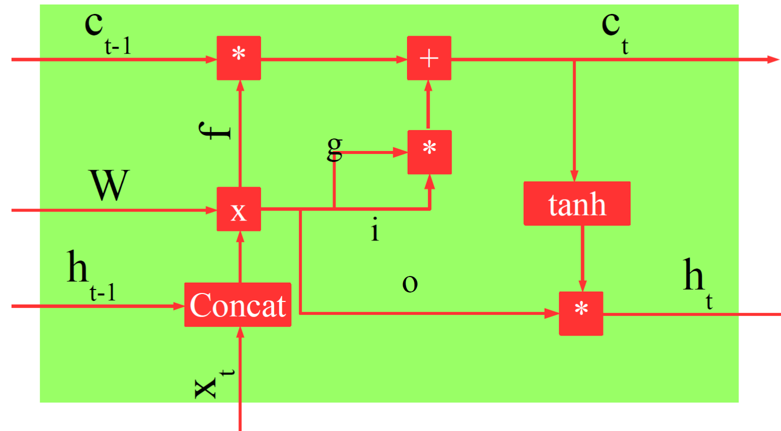
5.6. ábra. *A Backpropagation through time (BPTT) algoritmus elve. Amikor a háló végén játunk, csak a piros kerettel jelölt részig végezzük el a visszaterjesztést, különben a probléma mérete a végtelenségig növekedne.*

5.2.1. LSTM

A másik fontos különbség, hogy a kibontott sokrétegű háló esetén minden réteg súlymátrixa azonos (hiszen valójában egyetlen visszacsatolt rétegről van szó). Ez azt jelenti, hogy amikor a láncszabály segítségével a deriváltakat előállítjuk, akkor az egyes rétegek deriváltjainak egy hosszú szorzatát kell kiszámolnunk. Ebben az esetben azonban a szorzat minden eleme azonos, vagyis valójában egy hatványról beszélhetünk. Az pedig könnyen beláthatjuk, hogy a gyakorlatban bármilyen szám vagy mátrix sokadik hatványa vagy nulla vagy végtelen, kivéve, ha az a szám pontosan 1. Ebből következik, hogy egy visszacsatolt cella gradiensei könnyedén eltűnnek, vagy „felrobbannak”, ami a tanítást ellehetetleníti.

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

Az LTSM cella működése során az egyes kapuk hatása nélkül a c cella állapot korábbi értéke változás nélkül átíródik az aktuális állapotba, így a kettő közötti derivált pontosan egy. A cella



5.7. ábra. *Az LSTM cella felépítése.*

állapot értéke azonban még az egyes kapuk hatására módosulhat. Az első ilyen kapu a felejtés kapu f, amely egy a cella állapottal azonos méretű vektor, melynek minden eleme nulla és egy között van a szigmoid nemlinearitás hatására. A cella állapot vektorát ezzel a vektorral elemenként megszorozva a cella állapot bizonyos részleteit elfelejtjük.

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

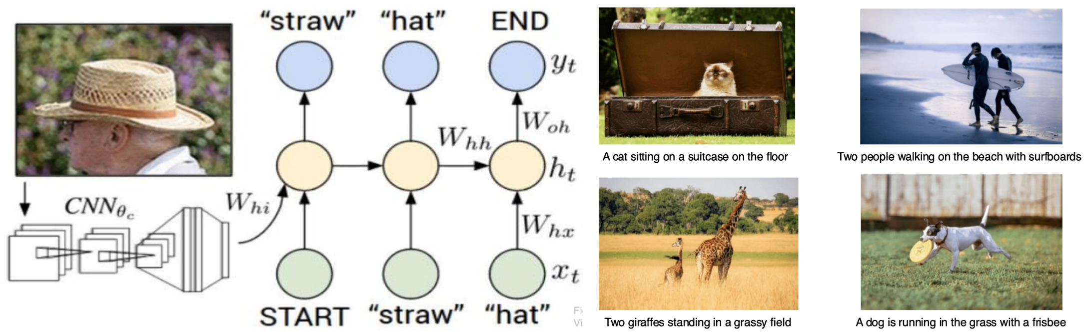
$$\begin{gathered}\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \\ c_t = f * c_{t-1} + i * g \\ h_t = o * tanh(c_t) \end{gathered} \tag{5.2}$$

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

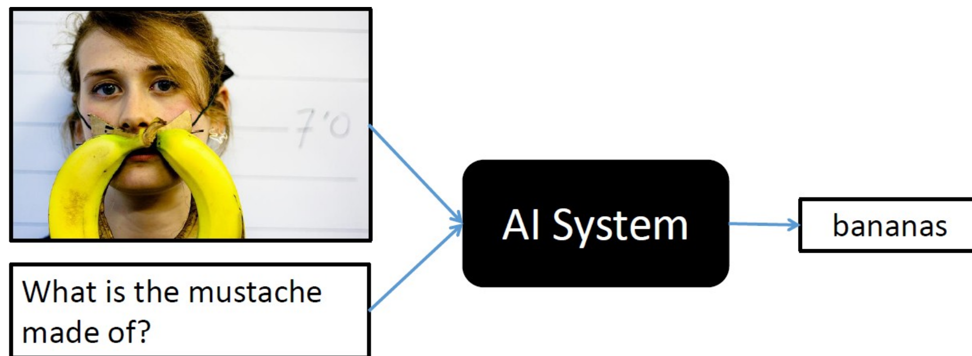
5.2.2. Alkalmazási példák

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

A visszacsatolt hálózatok egy rendkívül érdekes alkalmazása az úgynevezett puha figyelem modell megvalósítása. Ennek során egy visszacsatolt cella a kép tartalma alapján a kép egyes részeihez



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



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

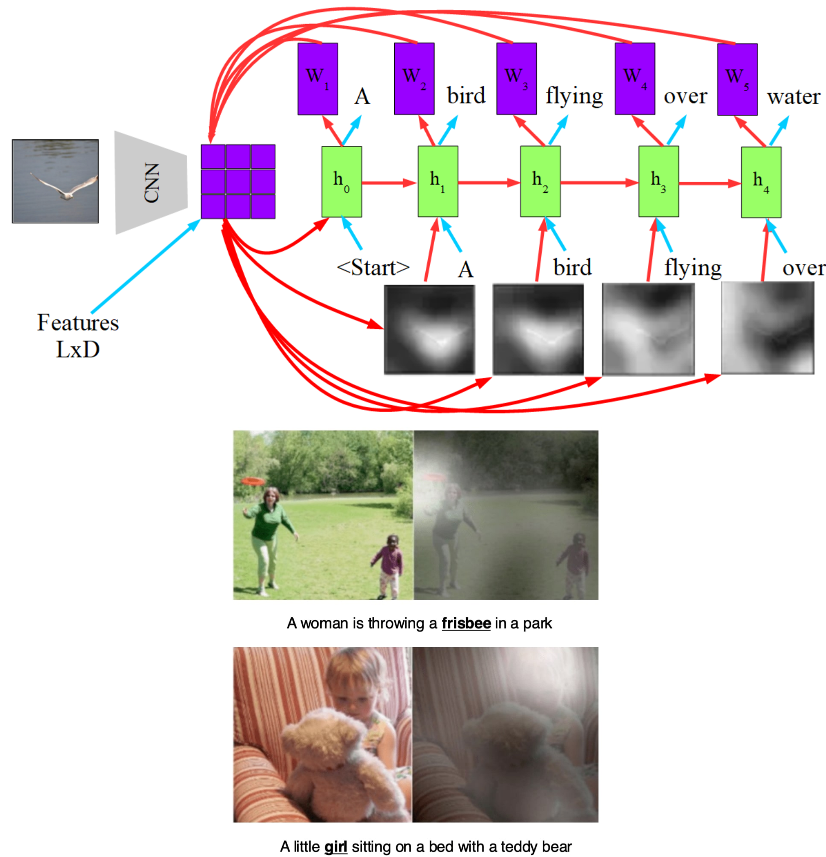
kiad egy-egy súlyt és az adott lépésben egy egyes részeket ezekkel a súlyokkal veszi figyelembe. A működés során a cella minden egyes lépésben új súlyokat generál, így folyamatosan változik, hogy a hálózat a kép melyik területeire összpontosít. Képek feliratozásakor megfigyelhető, hogy a mondat generálása során az egyes szavak kiadásának pillanatában a háló a képen pont oda figyel, ahol az adott szónak megfelelő tárgy található.

5.3. Transzformerek

5.3.1. Figyelem mechanizmus

A korábban ismertetett fejlettebb módszerek ugyan kezelik a hagyományos rekurrens architektúrák numerikus problémáit, azonban nem képesek ezt teljes mértékben megoldani. Ennek következményeképp az LSTM/GRU cellák sem bonthatók ki akármilyen mélységben, ami gátat szab az igazán hosszú időbeli függések megtanulásának. Erre a problémára igyekszik megoldást adni az ún. Önfigyelem (Self-Attention) réteg. A réteg alapelve röviden az, hogy egy bemeneti sorozat elemei között páronként egy figyelem súlyt számol, és az egyes elemek ennek a súlynak függvényében hatnak egymásra. A réteg kimeneteként is egy sorozatot állít elő, amelynek elemei a bemeneti sorozat elemeinek súlyozott kombinációi lesznek.

A réteg az alábbi módon működik: Először bemeneti sorozat minden elemére külön előállítunk három értéket. Ezeket key (kulcs) query (kérés) és value (érték) elemeknek hívjuk. Mindhárom értéket egy-egy külön lineáris réteg állítja elő, melyeket a bemeneti sorozat elemeire egymástól függetlenül futtatunk. Ezt követően a key és query értékeket páronként skalárisan szorozzuk, így előáll egy súlymátrix, melyben az i, j-edik elem a bemeneti szekvencia megfelelő elemei közti hasonlóságot fejezi ki. fontos, hogy felhasználás előtt ezt a mátrixot egy normalizációs tényezővel megszorzunk, mivel a skaláris szorzat eredményének szorzása az összeszorzott vektorok dimenziójának gyökével arányos. Ezzel a normálással a réteget a vektorok dimenziószámára invariánssá tesszük, majd ezután a szoftmax függvényt alkalmazzuk a hasonlósági mátrix soraira. Ennek ered-



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

ménye, hogy a mátrix minden eleme a $[0,1]$ tartományban található, valamint minden sor összege 1 lesz.

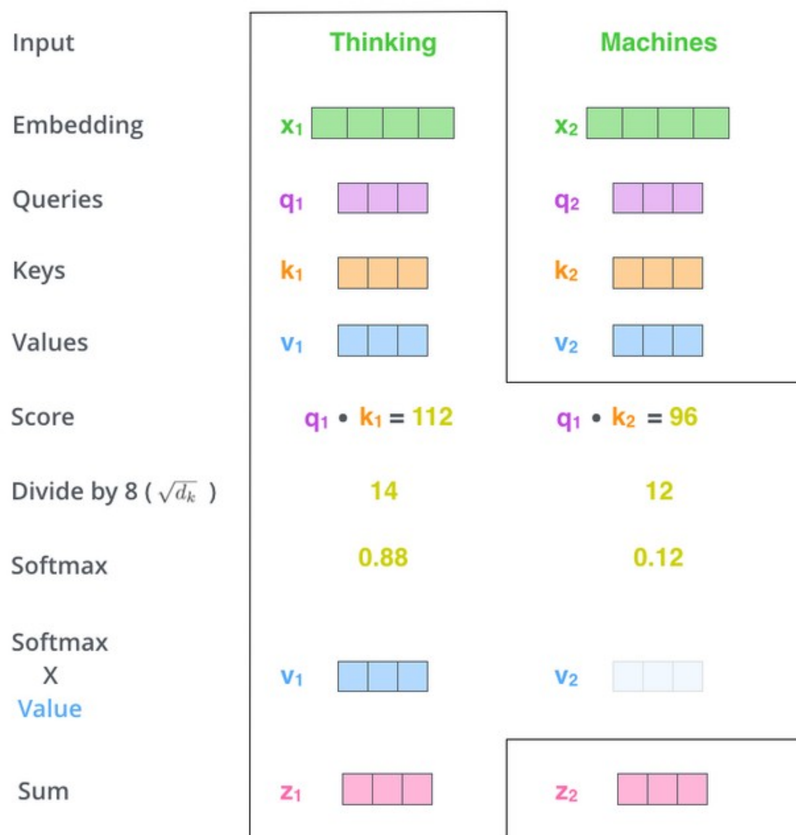
Ezt követően a sorozat minden eleméhez egy kimenetet számolunk, a kezdetben számolt value vektorok súlyozott összegzésével, ahol a súlyok a súlymátrix az adott bemenethez tartozó sorából jönnek. A self-attention réteg képlete mátrixos formában is kifejezhető. az alábbi módon:

$$Y = SoftMax(\frac{QK^T}{\sqrt{(d_k)}})V, \tag{5.3}$$

ahol Q, K és V a query, key és value vektorokat tartalmazó mátrixok, d_k pedig ezen vektorok dimenziója.

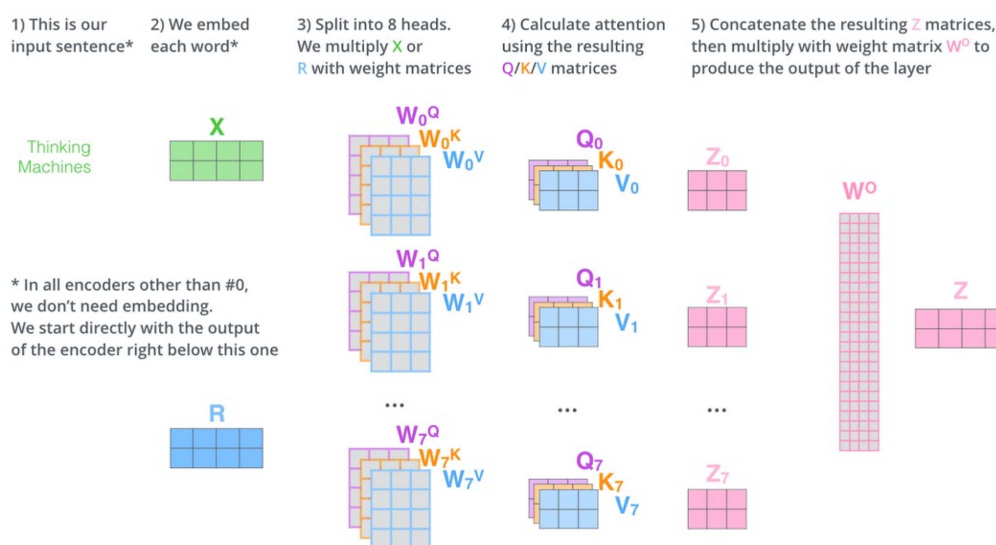
Érdemes megjegyezni, hogy az így kapott réteg a jelen formájában teljesen invariáns a bemeneti elemek sorrendjére, ami sorozatok esetében nem feltétlenül szerencsés. Éppen ezért a gyakorlatban nem közvetlenül a bemeneti sorozat elemeit, hanem azoknak valamilyen reprezentációját (embedding), adjuk a réteg bemenetére, és ez a reprezentáció már pozíció kódot is tartalmaz(hat). Ezek a pozíció kódok lehetnek előre meghatározott értékek, amelyek a sorozaton belüli elhelyezkedést kódolják; Erre gyakori különböző frekvenciájú szinusz és koszinusz kódokat alkalmazni. Valamelyest jobb teljesítmény érhető el azzal, ha a pozíció kódolásra használt elemek a háló tanulható paraméterei, ez azonban csak akkor alkalmazható, ha a réteg bemenetére adott szekvencia hosszára előzetes felső korlát mondható.

További fontos tulajdonság, hogy egy Self-Attention rétegnek akár több "feje" is lehet, melyek közül mindegyik saját key, query és value elemeket állít elő. A több fejből álló (Multi-head) attetion réteg külön kimeneteit először konkatenáljuk, majd egy végső lineáris réteg segítségével egyetlen kimeneti sorozattá konvertáljuk. A gyakorlatiban szinte mindig ilyen réteget használunk,



5.11. ábra. *A Self-Attention működésének alapelve.*

mivel ez egy adott sorozat elemei közti hasonlóságot több szempont szerint is ki tudja értékelni párhuzamosan, amely lényegesen növeli a réteg hasznosságát.



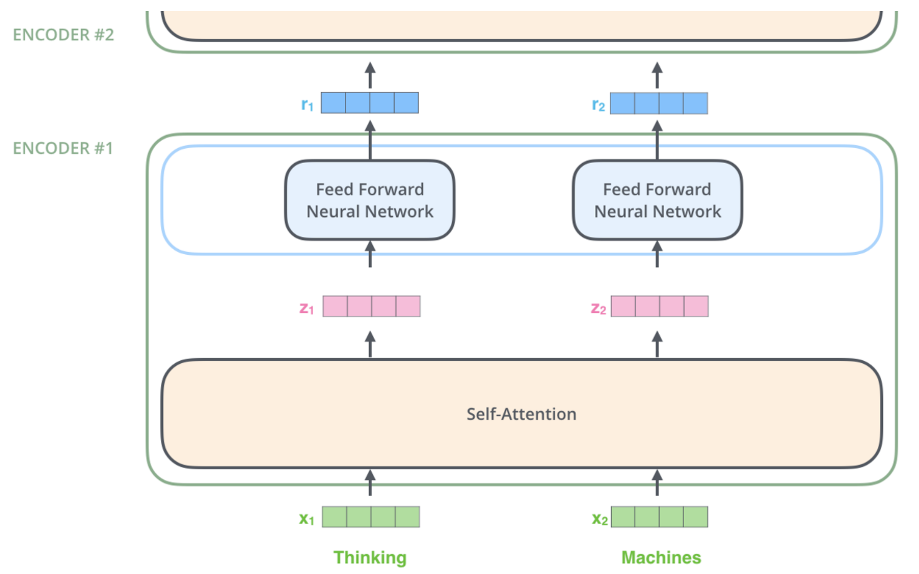
5.12. ábra. *A Multi-headed attention felépítése.*

Fontos végül megjegyezni, hogy a Self-Attention rétegből a Self kifejezés elhagyható, ekkor a réteg két különböző sorozat közti figyelmet fogja számolni. Ezt a megoldást Cross-Attention rétegnek nevezzük. Ebben az esetben a Source-nak nevezett sorozat elemeiből számítjuk a key és value elemeket, míg a Target-nek nevezett sorozat elemei adják a query értékeket. A réteg kimeneti

sorozata ebben az esetben a Target sorozat elemszámával fog megegyezni. Ily módon a korábban ismertetett Self-Attention elképzelhető, mint az a speciális eset, amikor a Source és a Target szekvenciák megegyeznek.

5.3.2. A transzformer architektúra

Az így megalkotott Multi-Head Attention (MHA) réteg felhasználható egy úgynevezett Transzformer encoder blokk megépítésére. Ezt úgy tesszük, hogy az encoder bemenetére adott szekvencia elemeit beadjuk egy MHA rétegnek, majd a kimeneti szekvencia elemeket egy két lineáris rétegből álló Multi-Layer Perceptron (MLP) hálónak adjuk át, amely ezeket az elemeket függetlenül dolgozza fel. Ebben az MLP hálózatban általában valamilyen nemlinearitást (célszerűen a ReLU valamilyen változatát), illetve Layer Normlaizációt alkalmazunk.



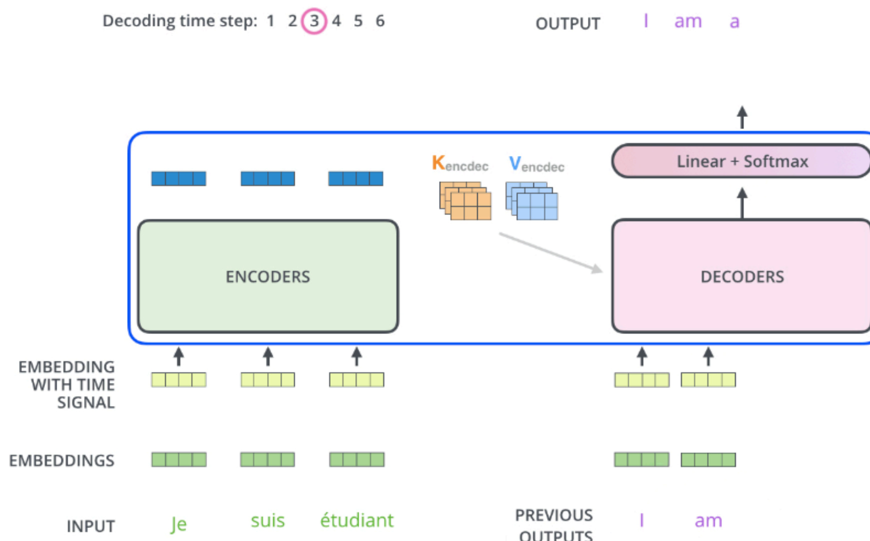
5.13. ábra. *Az encoder felépítése.*

Az így kapott encoder blokkokat egymás után fűzve kaphatunk egy transzfomer encodert, mely a bemenetére kapott szekvencia minden eleméhez készít egy olyan absztrakt leírót, amelybe a szekvencia többi eleméhez való kapcsolat is valamilyen szinten le van írva. Amennyiben a szekvencia minden eleméhez, vagy akár az egész szekvenciához egyetlen kimenetet szeretnénk generálni, akkor ez az encoder által generált jellemzőkön végrehajtott egyszerű osztályozó réteg segítségével megtehető.

Azonban amennyiben a bemeneti szekvenciához egy olyan kimeneti szekvenciát szeretnénk generálni, amelynek hossza nem feltétlenül egyezik meg, akkor szükséges egy decoder hálórész alkalmazása is. A decoder blokkok az encoder blokkokal megegyez módon épülnek fel, azonban nem self-, hanem cross-attention elemeket tartalmaznak, ahol a query a decoder bemenetéről, a key és a value pedig az encoder által kiszámolt jellemzőkből származik. Ennek alapján a decoder hálórészt iteratív módon futtatjuk: először egy START query-t adunk be neki, majd minden iterációban az előző körökben generált kimenetekhez tartozó query-ket is beadjuk neki, hasonlóan a képfeliratozásnál használt megoldásokhoz.

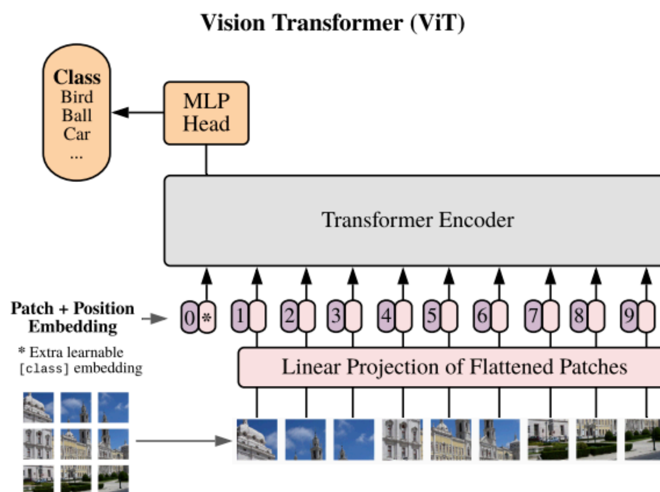
5.3.3. A Vision Transformer

Fontos megjegyezni, hogy habár a transformert elsősorban szöveges információk feldolgozására fejlesztették ki, nem sok idő kellett ahhoz, hogy a képi bemenetek feldolgozására is felhasználják. Ehhez először a képet szekvenciává kellett alakítani, amit úgy tesznek meg, hogy a képet $N \times N$-es rácsra osztjuk fel, és ezeket a patch-eket tekintjük egy szekvencia elemnek. Ahhoz, hogy ezek



5.14. ábra. *A Transzformer felépítése.*

Lineáris rétegeknek beadhatók legyenek, a patchekben lévő pixeleket egymás alá téve vektorizáljuk, 2D-ben pozíció enkódoljuk, majd ezekből készítjük a Q, K és V elemeket. Innentől a Vision és a hagyományos transzformerek működése megegyezik.



5.15. ábra. *A Vision Transzformer felépítése feliratozás esetében.*

Érdemes megjegyezni, hogy amennyiben a ViT architektúrát egyszerű osztályozásra szeretnénk használni, akkor a decoder rész elhagyható. Ebben az esetben az osztályozó kimenet bármelyik patchhez tartozó kimeneti elemre ráilleszthető, a gyakorlatban mégis szokás ehez egy extra query bemenetet adni a hálónak, amit class tokennek hívunk. Ez egy, az egy patchben lévő pixelek számával megegyező méretű paramétervektor, melynek értéke a tanítás közben a gradiens módszer segítségével tanulható.

Habár a transzformerek hatalmas előrelépést hoztak a természetes nyelvfeldolgozás (NLP) terén, a látásban nem adtak egyértelműen jobb eredményt a konvolúciós architektúráknál, ugyanis lényegesen nagyobb paraméterszámuk, illetve a különböző induktív bias-ok (transzláció invariancia, térbeli szomszédosság fontossága) hiányoznak. Éppen ezért tanításuk lényegesen hosszabb és több adatot igényel. Ennek kiküszöbölésére alkották a Data Efficient Image Tansformer (DeIT) archi-

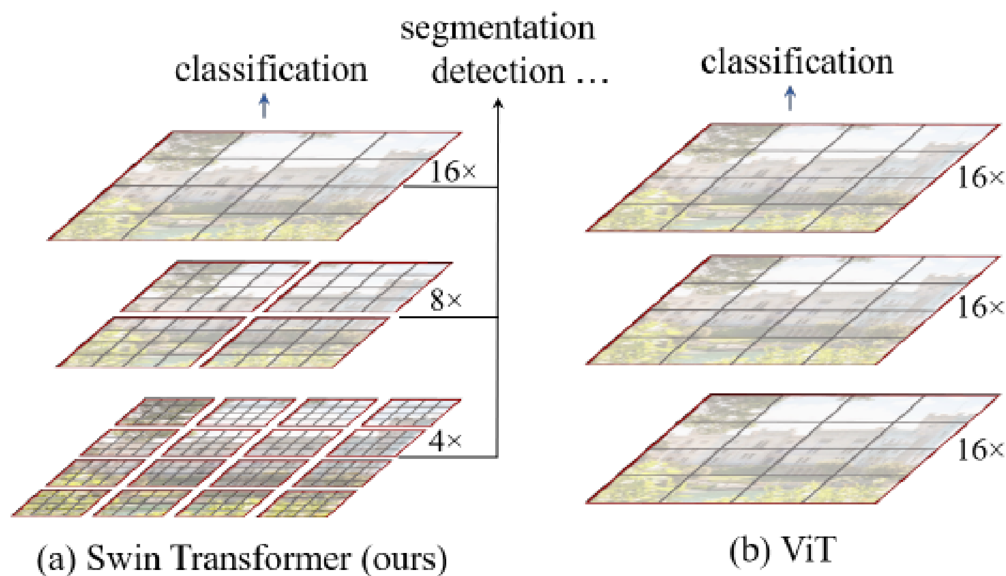
tektúrát, amelyben a class token mellé egy másik osztályozó kimenetet generáló, ún. disztillációs tokent is beadnak, és ezt a kimenetet egy másik már feltanított hálózat segítségével tanítják.

Ez a trükk a Knowledge Distillation kutatási területéről származik, melynek lényege, hogy egy már feltanult tanító (teacher) háló, az éppen tanuló (student) háló konvergenciáját a saját outputjai alapján segíti. Ez azét működik, mivel a háló a ground truth tanító címkéken kívül egy sokkal informatívabb visszajelzést is kap (a teacher háló kimeneti valószínűségei, esetleg feature-ei), aminek alapján sokkal könnyebben tud tanulni. Ez a tématerület azonban rendkívül mély és összetett, jelen tárgyban így csak említés szintjén foglalkozunk velük.

A SWin Transformer

A Vision Transformereknek azonban több fontos hátránya is van a konvolúciós architektúrákkal szemben. Ezek közül az egyik, hogy a self-attention mátrixának kiszámolása a bemeneti sorozat hosszával négyzetesen arányos memória és számításigényű, ami rendkívül pazarlónak számít. Ez lényegesen limitálja azt, hogy egy patch hány pixelből állhat, ugyanis túl kicsi patch esetén ezek száma túl nagyra nőne. Ennek következtében a vision transformerek kicsi, lokális jellemzőket nehezen detektálnak.

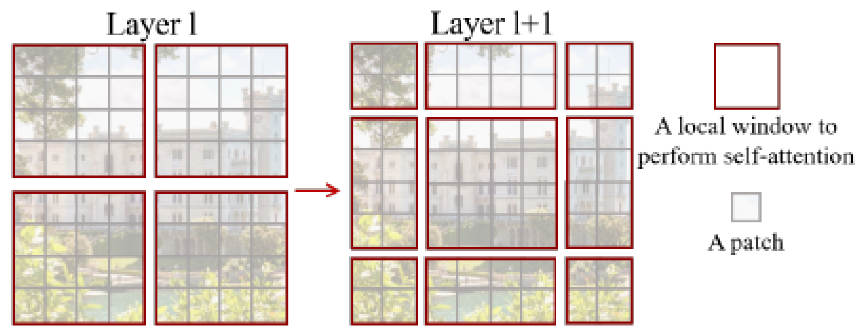
Ezt a problémát oldja fel a Shifted-Window (SWin) Transformer, amely a konvolúciós architektúrákhoz hasonló módon több szintre osztja a feldolgozást, ahol más skálák mentén vizsgálja a képet. A legalsó szinten a patchek mérete kicsi, azonban a self-attention-t nem számoljuk ki minden lehetséges patch-pár között, hanem csak egy (általában 4×4-es) lokális ablakokban, ezzel jelentősen csökkentve a memória- és számításigényt. A következő szinteken a patch és a lokális ablakok mérete egyre nő, ezáltal megvalósítva egy tipikus leskálázó jellegű encoder architektúrát. Ez lehetőséget nyit az encoder blokkok szélességének/csatornaszámának növelésére is, amely hagyományos transformerekre nem jellemző.



5.16. ábra. *A SWin transformer elve.*

Az elgondolásnak azonban van egy buktatója, mégpedig az, hogy az eltérő lokális ablakokban lévő, de egyébként szomszédos patchek között nincs információáramlás, így ha egy képjellemző pont erre a határra esik, akkor a hálózat csak nagyon nehezen tudja ezt detektálni. Éppen ezért egy adott szinten nem egy, hanem két egymást követő transzformer encoder blokk található, melyek közül a második a lokális ablakokat fél ablakmérettel elcsúsztatva használja. ezzel a módszerrel minden szomszédos patch között történik valamilyen infromációáramlás minden szinten.

A SWin transformerek már képesek voltak konvolúciós hálózatokat is túlszárnyaló, state-of-the-art eredmények elérésére, bár a két architektúra között jelenleg is szoros verseny folyik, e sorok olvasásának pontos idejétől függően éppen más hálótípus lehet az élen.



5.17. ábra. *A Shited-Window blokkok elve.*

További Olvasnivaló

[1] Jeff Heaton. „Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning". *Genetic Programming and Evolvable Machines* 19.1-2 (2017. okt.), 305–307. old. DOI: 10.1007/s10710-017-9314-z. URL: https://doi.org/10.1007%2Fs10710-017-9314-z.

[20] Andrej Karpathy és tsai. „Large-Scale Video Classification with Convolutional Neural Networks". *2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2014. jún. DOI: 10.1109/cvpr.2014.223. URL: https://doi.org/10.1109%2Fcvpr.2014.223.

[21] Joe Yue-Hei Ng és tsai. *Beyond Short Snippets: Deep Networks for Video Classification*. 2015. eprint: 1503.08909. URL: http://www.arxiv.org/abs/1503.08909.

[22] Shuiwang Ji és tsai. „3D Convolutional Neural Networks for Human Action Recognition". *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.1 (2013. jan.), 221–231. old. DOI: 10.1109/tpami.2012.59. URL: https://doi.org/10.1109%2Ftpami.2012.59.

[23] Karen Simonyan és Andrew Zisserman. *Two-Stream Convolutional Networks for Action Recognition in Videos*. 2014. eprint: 1406.2199. URL: http://www.arxiv.org/abs/1406.2199.

[24] Philippe Weinzaepfel és tsai. „DeepFlow: Large Displacement Optical Flow with Deep Matching". *2013 IEEE International Conference on Computer Vision*. IEEE, 2013. dec. DOI: 10.1109/iccv.2013.175. URL: https://doi.org/10.1109%2Ficcv.2013.175.

[25] Sepp Hochreiter és Jürgen Schmidhuber. „Long Short-Term Memory". *Neural Computation* 9.8 (1997. nov.), 1735–1780. old. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162%2Fneco.1997.9.8.1735.

[26] Kelvin Xu és tsai. *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*. 2016. eprint: 1502.03044. URL: http://www.arxiv.org/abs/1502.03044.

[27] Ashish Vaswani és tsai. *Attention Is All You Need*. 2017. eprint: 1706.03762. URL: http://www.arxiv.org/abs/1706.03762.

6. fejezet

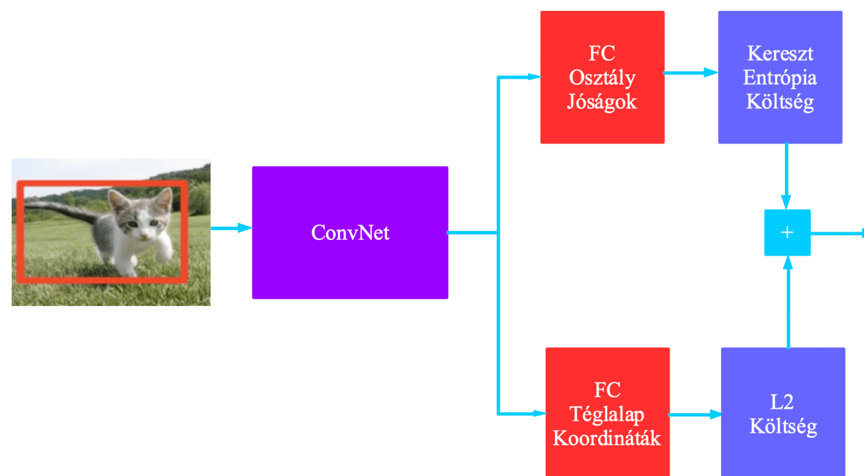
Detektálás és követés

6.1. Detektálás

Jelen fejezetben az egyik legfontosabb magas szintű tématerülettel, a detektálással foglalkozunk. Ennek során valamivel egyszerűbb a kinyert jellemző (általában befoglaló téglalapok), azonban megoldható az egyes objektumok különválasztása is.

6.1.1. Lokalizáció

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



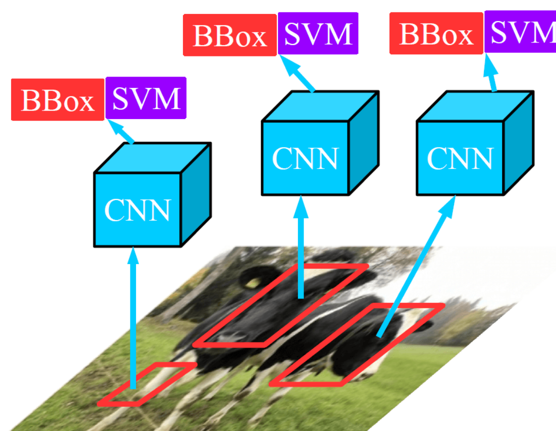
6.1. ábra. *A lokalizációt megvalósító architektúra.*

6.1.2. Régió-CNN

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

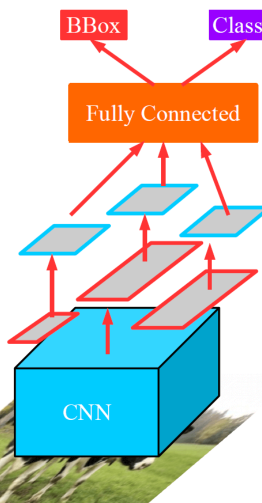
megalkotnunk. Természetesen az objektumok számára egy durva felső becslést adhatunk, így készíthetnénk egy olyan konvolúciós hálót, aminek pontosan ennyi különböző osztályozó és téglalap becslő kimenete van. Ez azonban N maximális objektum és C osztály esetében $N \times (C + 4)$ kimenetet jelentene,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

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



6.2. ábra. *A R-CNN architektúra.*

Az R-CNN módszer egyik legfontosabb hátránya, hogy az összes régiójavaslaton külön-külön futtatjuk le a neurális hálót, ami pazarlás. A módszer egy továbbfejlesztése, a FastR-CNN az egész képen futtat egy csak konvolúciós és leskálázó rétegekből álló hálót, majd az ez által elkészített aktivációs térképen keres régiójavaslatokat. Ezt követően ezeket a javaslatokat egy speciális pooling művelet segítségével azonos méretűre hozza (a hagyományos pooling műveletek egy adott faktorral skáláznak). Ezt követően a régió javaslatokon már csak egy kis méretű, csak lineáris rétegekből álló hálót futtat, amelyek az osztály és a téglalap becslését végzik (7.15 ábra). Ez a módszer az eredeti R-CNN módszerhez képest 10-20-szor gyorsabban működik.



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

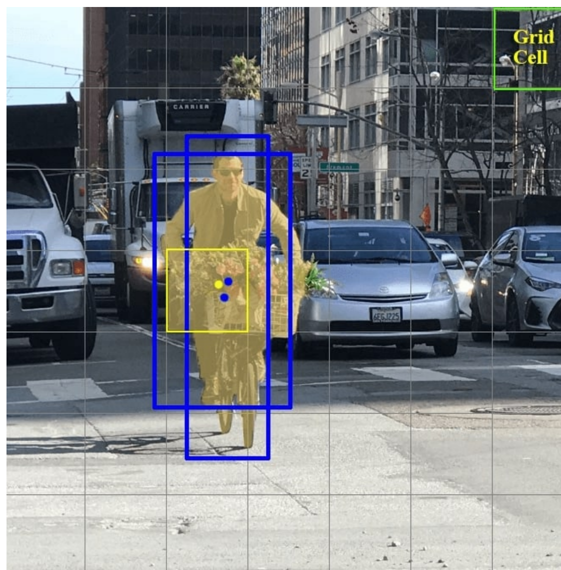
A FastR-CNN működésének a leglassabb része a régiójavaslatok előállítása, ami a futási idő 90%-

át teszi ki. Éppen ezért megalkottak még egy továbbfejlesztett változatot, ami a régiójavaslatok előállítását is egy RPN (Region Proposal Net) nevű neurális háló segítségével végzi el. Ez a háló a kezdeti konvolúciós rész által előállított aktivációs térképből állít elő fix számú régió javaslatot, amelyek mindegyikét binárisan osztályozza (objektum/nem objektum). Erre azért van szükség, mert a fix számú régió kimenet miatt a háló fixen ennyi régiójavaslatot tesz. A fő detektáló háló tanítása mellett az RPN hálót is arra tanítjuk, hogy a régiók befoglaló téglalapját és „objektumszerűségét" minél nagyobb pontossággal találja el. Ez a módszer a FastR-CNN megoldáshoz képest egy újabb tízszeres gyorsítást eredményez.

6.1.3. YOLO

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

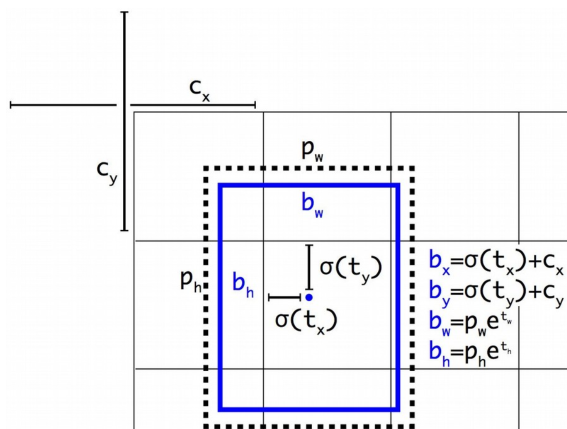
A végső becsléshez a YOLO a képet egy NxN-es rács segítségével felosztja, és minden cellából B darab objektumjelölt téglalapot becsül. Minden téglalaphoz tartozik egy C kimenetű osztályozó, valamint bináris osztályozó is, amely az adott téglalapba eső képrészlet „objektumszerűségét" adja meg. Így minden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



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

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

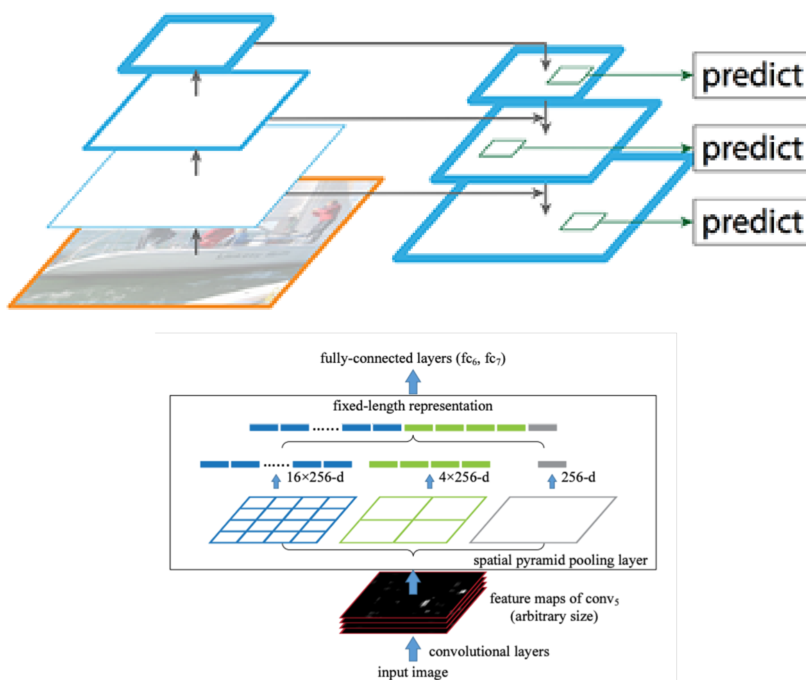
A YOLO-nak több változata is létezik, az anchor boxok használatát például a második verzió vezette be. A harmadik verzió a detektálást három különböző skálafaktor mellett is elvégzi, amihez az úgynevezett Feature Pyramid Network (FPN) modult alkalmazza. Ez egy olyan hálózatrész,



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

mely több egymás utáni felskálázó részt tartalmaz, és ezekhez még a háló elülső leskálázó részéből származó jellemzőket is előrecsatolja. Ennek érteleme, hogy így előállnak a háló eredeti végén keletkező nagy absztrakciós szintű jellemzők, csak éppen nagyobb felbontásban, ami a pontos lokalizációs és a kis objektumok megtalálását is segíti.

Gyakori trükk még a Spatial Pyramid Pooling (SPP) nevű modul alkalmazása, amely gyakorlatilag több, egymással párhuzamosan alkalmazott, eltérő méretű pooling réteg alkalmazása. Ennek segítségével elő lehet állítani ugyanazt a képjellemzőt több különböző látómezővel, amely a különböző skálájú objektumok egyidejű detektálását könnyíti meg. Ezekkel a megoldásokkal a YOLO teljesítménye jelentősen javul különböző (de legfőképp kis) méretű objektumok pontos detekciója terén.

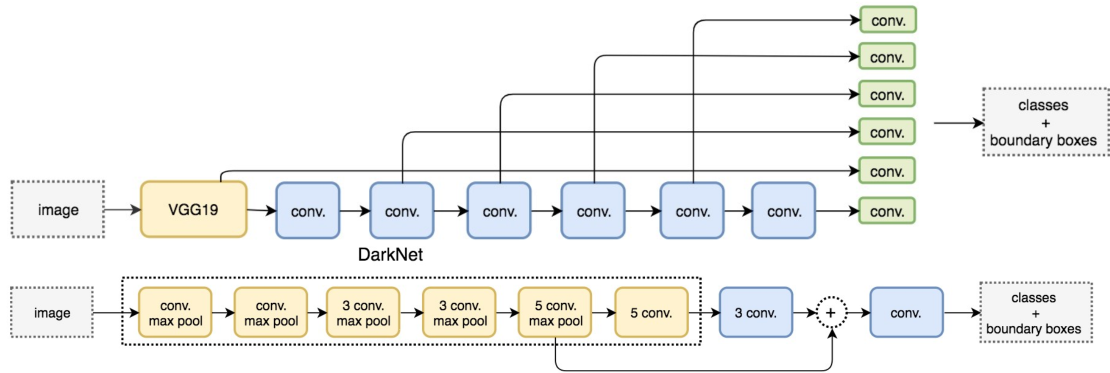


6.6. ábra. *Az FPN (felül) és az SPP (alul) modulok.*

Jelen sorok írásakor a YOLOv8-as változatnál tartunk, mely már nem csak objektumdetektálásra, hanem instance szegmentációra, illetve egyes objektumokhoz kulcspontdetekcióra is képes. Ezen felül ez a verzió a korábbiakhoz képest lényegesen fejlettem konvolúciós backbone részt kapott, illetve az adataugmentációs technikákat is nagymértékben fejlesztették. A YOLO legfőbb erénye a

régió alapú detektálással szemben, hogy rendkívül gyors, így megfelelő hardver használata esetén valósidejű működésre is képes, különösképp a TinyYOLO névre hallgató változata.

Érdemes még megemlíteni a Single-Shot Detector (SSD) névre hallgató algoritmust, amely a YOLO-hoz meglehetősen hasonló alapelven működik (szokás is a YOLO-szerű algoritmusokat általánosságban SSD detektoroknak hívni). Az egyik legalapvetőbb különbség, hogy az SSD egy előre tanított VGG háló konvolúciós részét használja, majd ehhez add hozzá további konvolúciós részeket. A YOLOv3-hoz hasonló módon az SSD is több skála mentén végez detektálást.



6.7. ábra. *Az SSD (felül) és a YOLO (alul) architektúra.*

6.1.4. Anchor Free

Érdemes megjegyezni, hogy a YOLO és más Single-Shit detektorok esetében nincs kizárva az, hogy egy objektumot egyszerre több kimenet is többé-kevésbé helyesen megbecsüljön, aminek eredménye sok egymással átlapoló téglalap lesz. Ennek elkerülése végett a detekciókon egy Non-Maximum Supression (NMS) nevű utófeldolgozó műveletet végzünk el, melynek során az egymással átlapoló és azonos osztályt predikáló téglalapok közül csak a legnagyobb konfidenciájút hagyjuk meg. Bár ez a művelet nem különösebben számításigényes, azonban nem párhuzamosítható, és CPU oldalon elvégzendő, ami azt jelenti, hogy a futási ideje (főleg egy kicsi YOLO háló esetében) mégis számottevő lehet.

Ennek elkerülésére megalkottak olyan, úgynevezett Anchor-Free módszereket, melyek a detekciókat egy másik elv mentén végzik. Ennek lényege, hogy a háló relatíve nagy felbontású rácson becsül minden cellához egy konfidenciát (hogy van-e ott objektum), illetve egy befoglaló téglalapot osztály jóságokkal együtt. Ezt követően a konfidencia térképen lokális maximumokat keresünk, mégpedig úgy, hogy először egy 3×3-as max szűrőt futtatunk a tömbön (ezt célszerűen egy 1-es stride-ú max pooling réteggel valósítjuk meg), majd a szűrt és az eredeti tömb elemei között egyenlőséget keresünk. Belátható, hogy mivel a max szűrés a lokális maximumokat "elkeni" ezért a szűrt kép csak a maximumok helyén lesz egyenlő az eredetivel. Ennek a megoldásnak a segítségével két rendkívül olcsó és triviálisan párhuzamosítható művelettel elkerülhetjük az NMS művelet végrehajtását.

Az Anchor-Free módszerek közül az egyik legismertebb a fenti sorokban bemutatott CenterNet, de létezik még ennek az ötletnek számos további variánsa. Érdemes megjegyezni, hogy ezek az architektúrák pontosság terén versenyeznek a YOLO-féle megoldásokkal, míg - főleg kis architetúrák esetében - általában gyorsabbak is.

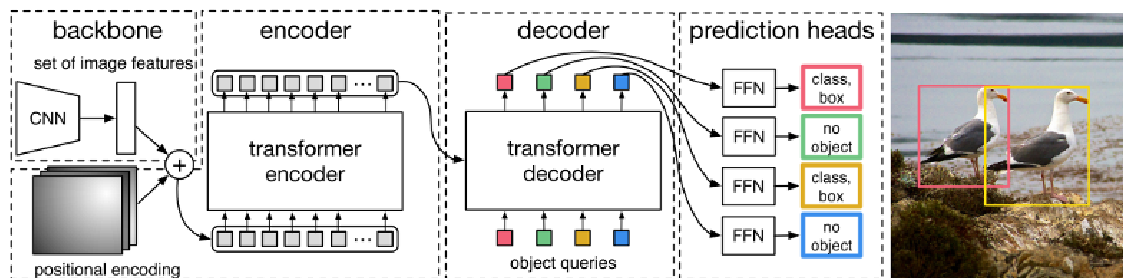
6.2. Detektálás Transformerek segítségével

Természetesen az utóbbi években számos megoldás született a korábbi fejezetben ismertetett Vision Transzformer architektúrák felhasználására a detektálás területén. Ezek közül az egyik gyakori megoldás, amikor a YOLO algoritmust nem egy konvolúciós, hanem egy Swin-Transzformeres backbone hálózattal használják. Ennek megértéséhez érdemes belátni, hogy a YOLO maga nem

egy neurális háló architektúra (maga az eredeti YOLO csupán egy egyszerű teljesen konvolúciós háló), hanem az annak végére tett predikciós és költségfüggvény modul. Ezt a modult gyakorlatilag minden különösebb megfontolás nélkül oda lehet illeszteni egy ViT kimenetére is, hiszen az sem csinál mást, mint a kép minden részéhez egy leíró vektort becsül.

Ezzel teljesen ellentétes logikát képvisel a DE:TR (Detection with Transformers), architektúra, amely meghagyja a transzformerekhez képest meglehetősen hatékony konvolúciós jellemzőkinyerő backbone részt, és magát a detekciós sémát valósítja meg transzformerek segítségével. Működése során a konvolúciós rész által szolgáltatott jellemzőket először enkódolja (a griden lévő pozíció alapján), majd egy encoder hálórésznek adja át. Ennek a kimenetét aztán beadja a transzformer decoder részének egyik bemeneteként.

A decoder másik bemenete egy előre meghatározott hosszúságú objektum query szekvencia, melynek értékei a háló tanulható paraméterei. Ez a tanulás természetesen a gradiens módszer segítségével történik, tekintve, hogy a query vektorok rajta vannak a számítási gráfon, így ez egyszerűen megolható. A mivel query szekvencia minden eleméhez egy kimenet fog tartozni (és minden egyes kimenet egy detekció lesz), ezért ennek hosszát úgy érdemes meghatározni, hogy biztosan több query legyen, mint ahány objektum egy képen előfordulhat.



6.8. ábra. *A DETR architektúra.*

Fontos tisztázni azt is, hogy a tanítás közben hogyan rendeljük hozzá a ground truth objektumokat az egyes kimenetekhez, ez ugyanis elengedhetetlen ahhoz, hogy elő tudjuk írni, hogy melyik kimenetnek mit kellett volna predikálnia. Ezt a DETR esetében úgy tesszük, hogy először minden predikciót és minden groung truth objektumot összehasonlítunk, és felírunk köztük egy jóság mércét, ami azt fejezi ki, hogy az adott predikció mennyire van közel a konkrét detektálandó objektumhoz. Ezt általában a téglalapok átfedése (illetve annak általánosított változata) alapján végezzük el. Ezt követően a hasonlóságok alapján egy optimális párosítási problémát kell megoldani, amelyet a Magyar algoritmus alapján végzünk el.

Habár ezzel a módszerrel a párosítást meg lehet oldani, azonban fontos megjegyezni, hogy a kezdeti teljesen véletlenszerű predikciók során a ground truth objektum hozzárendelése folyamatosan és drasztikusan változhat, ami jelentősen megnehezítheti a tanulást. Fontos továbbá megjegyezni azt is, hogy a DETR sokkal kevésbé hajlamos többszörösen detektálni ugyanazt az objektumot, de ez nem jelenti azt, hogy az NMS lépés teljes mértékben elhagyható.

6.3. Követés

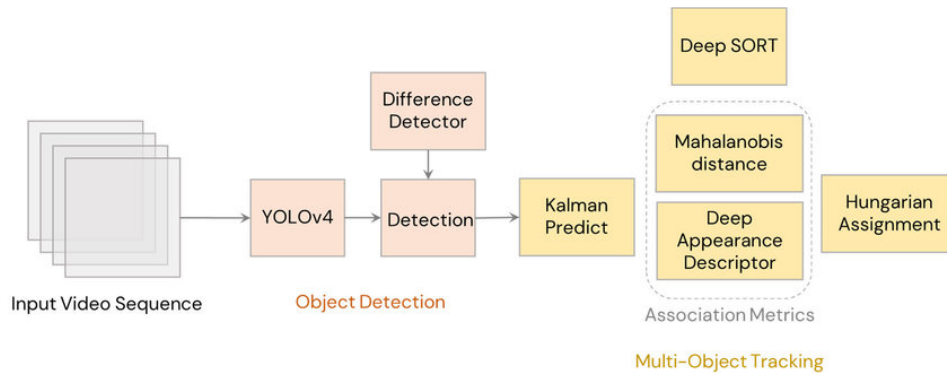
A detektáláshoz kapcsolódó, de annál egy fokkal nehezebb látási probléma az objektumkövetés. A nehézség oka, hogy nem elég csupán minden képkockán az objektumokat detektálni, hanem ezeket a különböző képkockák közt párosítani is kell. Az optimális párosítási probléma ugyan megoldható polinomiális időben a korábba említett Magyar algoritmus segítségével, azonban egyáltalán nem mindegy, hogy a párosításhoz használt hasonlóságokat mi alapján számoljuk. Ezen felül lehetséges a követést úgy is elvégezni, hogy az egyszer detektált objektumok elmozdulását próbáljuk csupán becsülni a következő képeken, ez azonban a relatív becslések hibáinak összegződése miatt egyre növekvő követési hibához, vagyis más néven drift-hez vezethet.

6.3.1. DeepSort

Az egyik legelterjedtebb követési algoritmus az úgynevezett DeepSort, mely egy YOLO hálózatot használ detektornak. Ezt követően megpróbálja a detekció következő pozícióját megbecsülni az eddigi ismert pozíciók alapján. Ehhez egy Kalman szűrő predikciós lépését használja fel (ez egy a szabályozástechnikában elterjedt módszer, amely egy rendszer lineáris állapotegyenlete alapján végez egy lépéssel előre tartó predikciót).

Ezt követően a előző és aktuális detektciók közötti hasonlóság kiszámolása következik, amelyet két módszerrel végez a DeepSort. Ezek közül az első a térbeli pozíciók közti Mahalanobis távolság számolása alapján történik. Ehhez azonban a DeepSort hozzávesz egy másik tényezőt; Ezt úgy teszi, hogy egy külön konvolúciós hálózat segítségével minden detektált objektumhoz elkészít egy absztrakt leíró jellemzőt (ez valamilyen N dimenziós vektor), és az ezek közti távolságot is kiszámítja. A végleges hasonlóság a két részelem súlyozott átlagaként adódik.

A végső párosítást eztán a Magyar módszer segítségével kapjuk. A DeepSort egyik nagy előnye, hogy a követéshez nem csak az egyes objektumok térbeli pozícióját, hanem a megjelenésbeli hasonlóságukat is felhasználja. Érdemes még megjegyezni, hogy a megjelenés leíróját előállító konvolúciós hálót egy trackelésre felcímkézett adatbázis segítségével taníthatjuk be a kontrasztív tanulás technikájával, mellyel egy későbbi fejezetben fogunk részletesen beszélni.



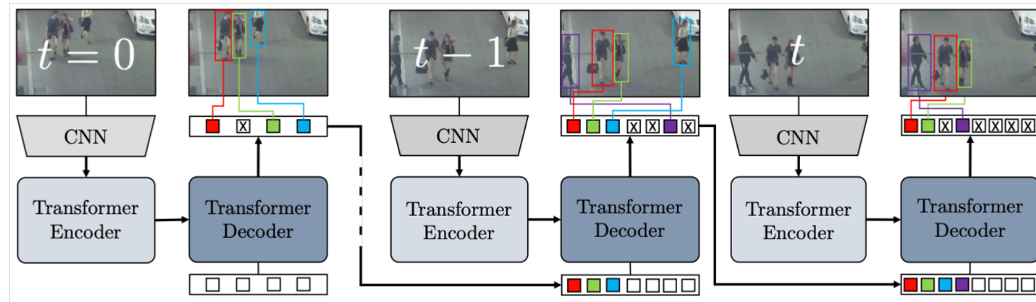
6.9. ábra. *A DeepSort architektúra.*

6.3.2. Követés Transzformerekkel

Felmerülhetett az eddig olvasottak alapján, hogy a Transzformer hálókban lévő Attention réteg (főleg, ha azt Cross-Attention módban használjuk) gyakorlatilag mintha a párosítások problémájára lett volna kitalálva. Hiszen pont azt csinálja, hogy a bemenetére adott két szekvencia elemei közt hasonlóságokat számol, majd a kimenetet ezek alapján állítja össze. Éppen ezért a Transzformerek követésre való felhasználása elég könnyen adódik az architektúrából.

Az első ilyen megoldás a TrackFormer módszer, amely gyakorlatilag a DETR hálót használja fel egyetlen apró, de jelentős módosítással. Az első képkockán a módszer működése megegyezik a DETR-el, tehát a kép egy konvolúciós enkóderen keresztülmegy, majd azt egy transzformer enkóder követi, végül pedig a háló az objektum query-kre adott válaszként detektál néhány objektumot. A következő képkocka sorsa ugyanez lesz, azonban azzal a fontos különbséggel, hogy a transzformer dekóder ekkor az előre megtanult objektum query-k mellé az előző képkocka detekcióit is megkapja extra track queryk formájában. Ehhez persze az architektúrán semmit nem kell változtatni, mivel a transzformerek invariánsak a bemeneti szekvenciáik hosszára.

A TrackFormer fontos ötlete abban teljesedik ki, hogy a kimenetén azt írjuk elő, hogy a más korábban megtalált objektumok a hozzájuk tartozó track querykből számolt kimeneten álljanak elő, míg az általános objektum query-k kizárólag az újonnan megjelent objektumok detektálására vannak. Ezzel a trükkel a TrackFormer megtanulja belül elvégezni a párosítási feladatot, így nincs



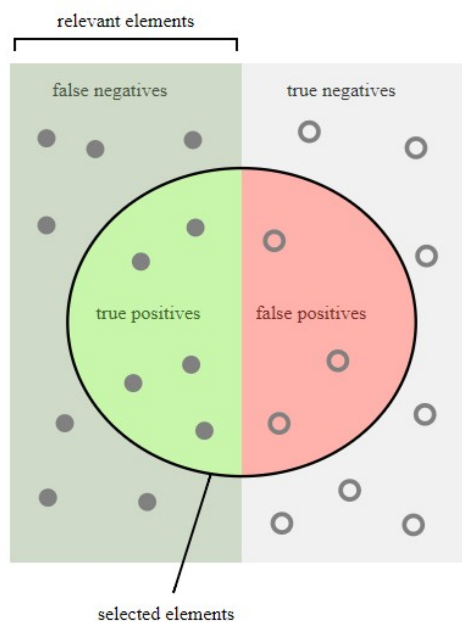
6.10. ábra. *A TrackFormer architektúra.*

szükség a Magyar algoritmus futtatására, illetve azt sem kell kitalálni, hogy mi alapján kell a hasonlóságot számolni, hiszen a TrackFormer ezt s megtanulja.

A TrackFormer egy fontos továbbfejlesztett változata a MOTR, amely annyiban különbözik, hogy a dekóder bemenetére adott query-k nem csak az objektum előző állapotát, hanem a teljes track history-t is elkódolják. Ezt a MOTR egy úgynevezett Query Interaction Module (QIM) segítségével végzi el, melynek lényege, hogy az aktuális detekciót a hozzá tartozó track query-vel összekombinálja, így összefűzve az aktuális információt az összes korábbival.

6.4. Fontos mérőszámok

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

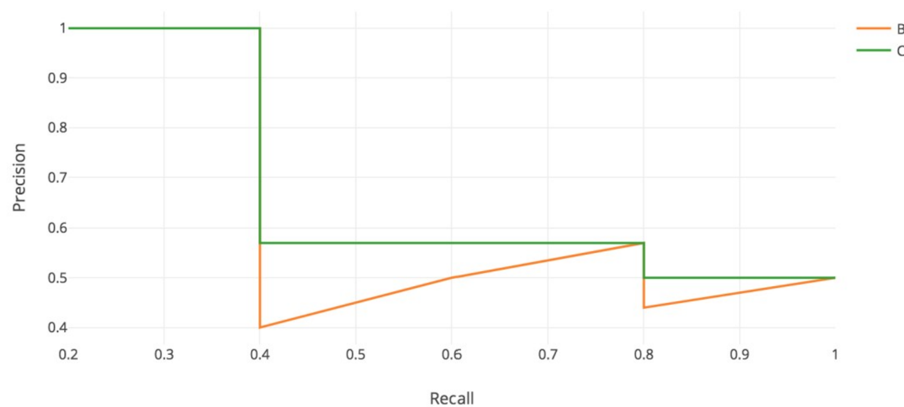


6.11. ábra. *A kétféle tévesztés lehetősége bináris osztályozás esetén.*

A pontosság azt fejezi ki, hogy a tanuló algoritmus által pozitívnak kiválasztott elemek (vagyis az éppen vizsgált osztályba tartozó) hány százaléka volt valóban pozitív. Az emlékezés pedig azt

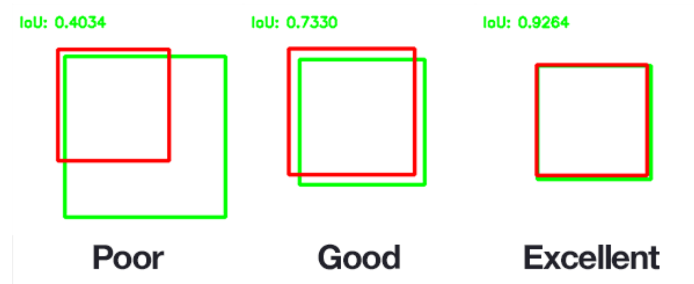
fejezi ki, hogy a ténylegesen pozitív elemek hány százalékát választotta az algoritmus pozitívnak. A jó osztályozáshoz ezeknek a mérőszámoknak együtt kell kielégítőnek lenniük. Érdemes még megjegyezni, hogy többértékű osztályozás esetében az úgynevezett keverési mátrixot szokás vizsgálni, amely voltaképpen ezen mérőszámok kiterjesztése. A recall és a precision értékek egymással valamelyest ellentétes mérőszámok, így gyakran ábrázoljuk a kettő közti kompromisszumok lehetőségét a recall-precision görbe segítségével.

$$\begin{aligned} Accuracy &= \frac{TP+TN}{Total} \\ Precision &= \frac{TP}{TP+FP} \\ Recall &= \frac{TP}{TP+FN} \end{aligned} \tag{6.1}$$



6.12. ábra. *Egy recall-precision görbe (mérésekkel közelítve).*

Detektálás esetén azonban nem csak az osztályozás, hanem a lokalizáció helyességét is szükséges mérni és értékelni. Erre az Intersection over Union (IoU) mérőszám használatos. Ahogy az a mérőszám nevéből is adódik ez a becsült és az igazi befoglaló téglalap metszetének területét osztja le az unió területével, így egy 0-1 közti mérőszámot kapunk a lokalizáció jóságára. Leggyakrabban a 0.5 vagy a 0.75 fölötti IoU érték esetén tekintjük a detekciót helyesnek.



6.13. ábra. *Az IoU mérőszám.*

$$IoU = \frac{True \wedge Predicted}{True \vee Predicted} \tag{6.2}$$

A fenti mérőszámokat összerakva megalkothatjuk a detektáló algoritmusok teljesítményét kompakt módon összefoglaló mérőszámot. Ehhez először kiszámoljuk az úgynevezett átlagos pontosságot (AP), ami voltaképp a recall-precision görbe alatti terület. Ezt természetesen téglalapos módszerrel közelítjük. Ezt követően az AP mérőszámot az összes osztályra kiszámolva és összeátlagolva megkapjuk a mean Average Precision, vagyis mAP értéket. A mAP értéket szokás több IoU küszöbérték mellett is megvizsgálni.

$$AP = \frac{1}{N} \sum_{i \in \{0.0 \ldots 1.0\}} P_i \tag{6.3}$$

További Olvasnivaló

[1] Jeff Heaton. „Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning”. *Genetic Programming and Evolvable Machines* 19.1-2 (2017. okt.), 305–307. old. DOI: 10.1007/s10710-017-9314-z. URL: https://doi.org/10.1007%2Fs10710-017-9314-z.

[14] Jonathan Long, Evan Shelhamer és Trevor Darrell. „Fully convolutional networks for semantic segmentation”. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015. jún. DOI: 10.1109/cvpr.2015.7298965. URL: https://doi.org/10.1109%2Fcvpr.2015.7298965.

[15] Liang-Chieh Chen és tsai. *DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs*. 2017. eprint: 1606.00915. URL: http://www.arxiv.org/abs/1606.00915.

[16] Shuai Zheng és tsai. „Conditional Random Fields as Recurrent Neural Networks”. *2015 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2015. dec. DOI: 10.1109/iccv.2015.179. URL: https://doi.org/10.1109%2Ficcv.2015.179.

[17] Shaoqing Ren és tsai. „Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (2017. jún.), 1137–1149. old. DOI: 10.1109/tpami.2016.2577031. URL: https://doi.org/10.1109%2Ftpami.2016.2577031.

[18] Joseph Redmon és tsai. „You Only Look Once: Unified, Real-Time Object Detection”. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016. jún. DOI: 10.1109/cvpr.2016.91. URL: https://doi.org/10.1109%2Fcvpr.2016.91.

[19] Kaiming He és tsai. „Mask R-CNN”. *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017. okt. DOI: 10.1109/iccv.2017.322. URL: https://doi.org/10.1109%2Ficcv.2017.322.

7. fejezet

Szegmentálás és videóanalízis

7.1. Szemantikus szegmentálás

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

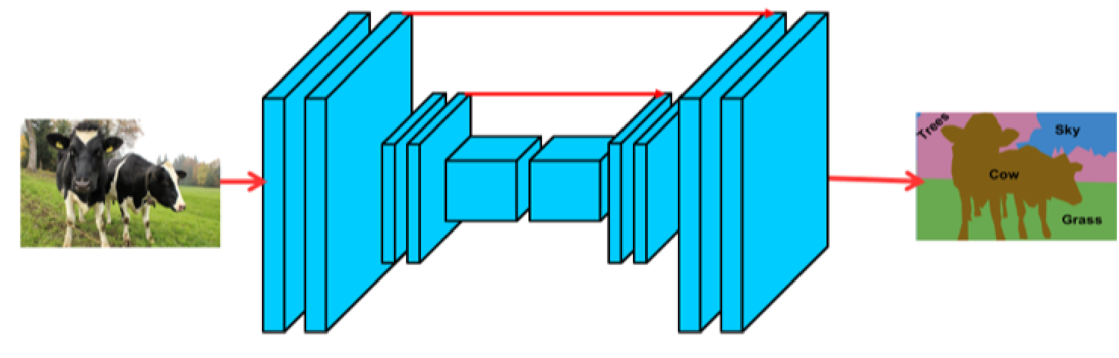
7.1. ábra. *A szemantikus szegmentálás feladata.*

7.1.1. Teljesen konvolúciós architektúra

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

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

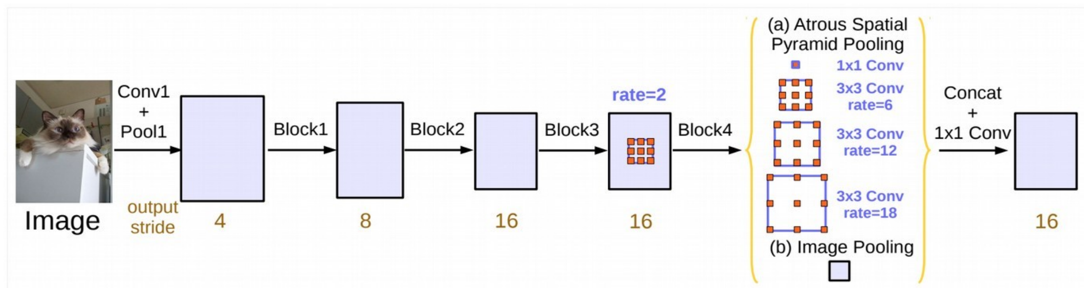
szintű jellemzők segítenek az osztályok határvonalának minél pontosabb meghatározásában, amik a leskálázás során elvesznek.



7.2. ábra. *Egy tipikus FCN architektúra.*

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

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



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

Érdemes még a hálók tanítása során alkalmazandó hibafüggvényről szót ejteni. A leggyakrabban használt és leginkább kézenfekvő választás az osztályozás során megismert kereszt-entrópia költség alkalmazása, pixelenként összegezve. Ennek hátránya, hogy kis méretű objektumok elvesztéséért relatíve kis büntetést kapunk, ezért a háló gyengébben fog teljesíteni ezekben az esetekben. Ennek a problémának az orvoslására bevezethetjük az úgynevezett Dice költségfüggvényt, amely az alábbi módon adódik:

$$\begin{gathered} Dice = \frac{2(Pred \wedge True)}{Pred \vee True} \\ Loss = 1 - Dice \end{gathered} \tag{7.1}$$

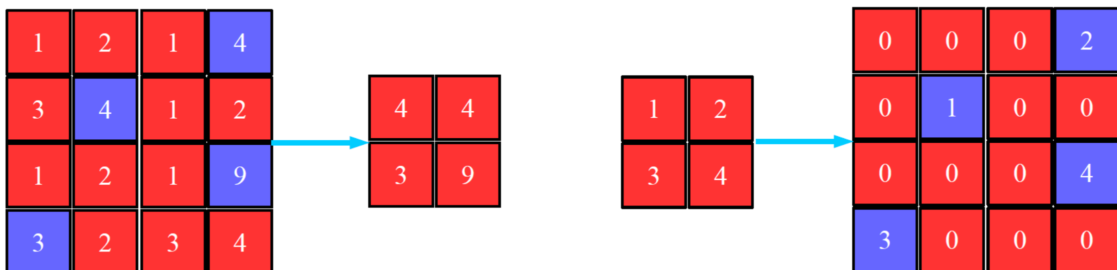
A gyakorlatban általában ennek a fuzzy osztályzásra kiterjesztett változatát alkalmazzuk, amely a Soft Dice Loss névre hallgat:

$$Dice = 1 - \frac{2\sum(y_p y_t)}{\sum y_p^2 + \sum y_t^2} \tag{7.2}$$

Ezeket osztályonként/objektumonként külön számolva és összeadva biztosíthatjuk a megfelelő tanítást.

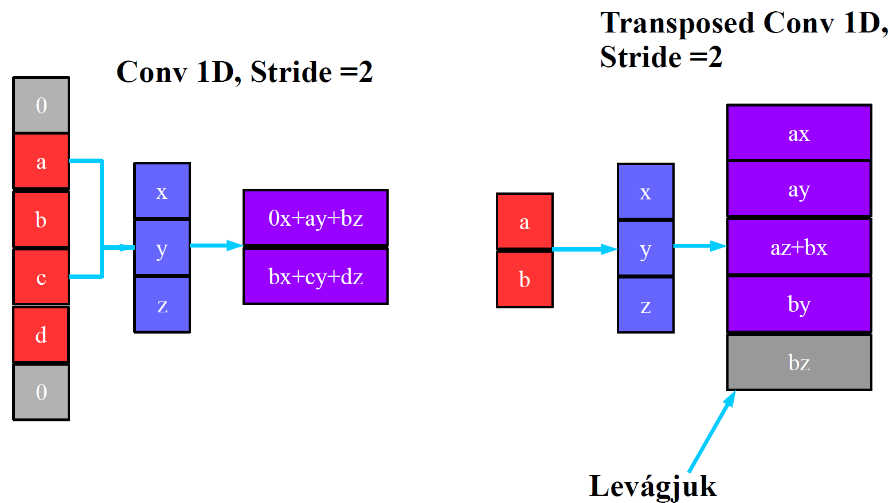
7.1.2. Felskálázás módszerei

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



7.4. ábra. *A max unpooling művelet.*

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



7.5. ábra. *A transzponált konvolúció 1D-ben.*

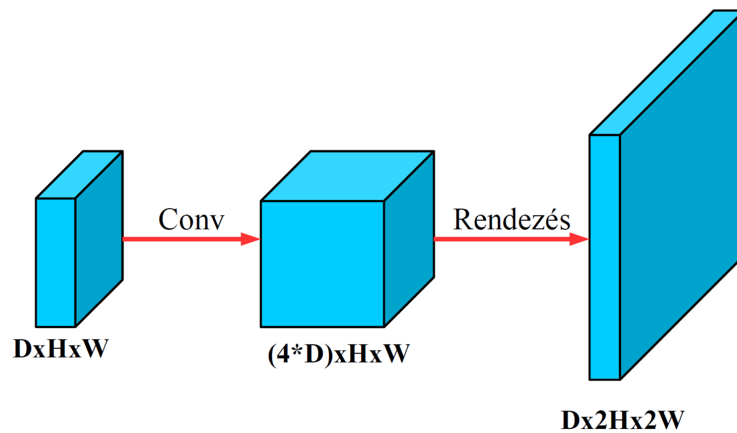
A transzponált konvolúció neve onnan ered, hogy a konvolúció leírható egy mátrixszorzással az alábbi módon:

$$\begin{pmatrix} x & y & z & 0 & 0 & 0 \\ 0 & x & y & z & 0 & 0 \\ 0 & 0 & x & y & z & 0 \\ 0 & 0 & 0 & x & y & z \end{pmatrix} \begin{pmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{pmatrix} = \begin{pmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + by \end{pmatrix} = Xa \tag{7.3}$$

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

$$\begin{pmatrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} ax \\ ay + bx \\ az + by + cx \\ bz + cy + dx \\ cz + dy \\ dz \end{pmatrix} = X^T a \tag{7.4}$$

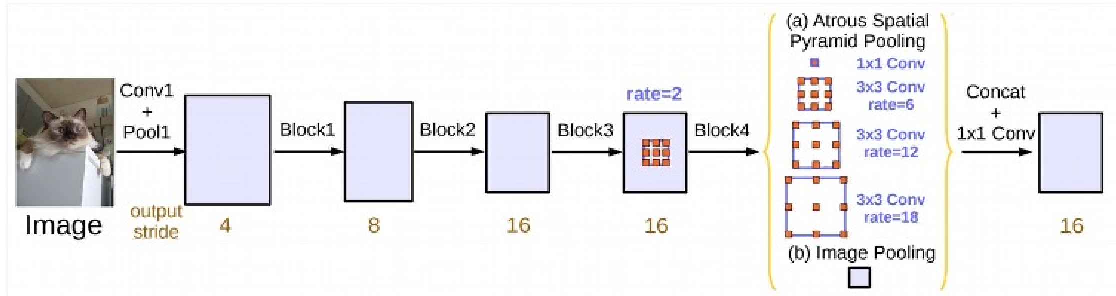
Létezik még egy harmadik elterjedt módszer is, ez a sűrű felskálázó konvolúció. Ennek lényege, hogy egy konvolúciós réteg segítségével az adott aktivációs térkép csatornáinak számát a négyszeresére növeljük, majd az így kapott tömböt átrendezzük úgy, hogy az aktivációs térkép térbeli méretei az eredeti kétszeresei legyenek, csatornának száma pedig egyezzenek meg az eredetivel. Ez a módszer szintén tanuló felskálázás, viszont több paraméterrel rendelkezik, így képes komplexebb transzformációk megtanulására lassabb működés árán.



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

A szegmentáló architektúrákban kifejezetten gyakori a dilatált, vagy más néven atrous konvolúciós kernelek alkalmazása. Ha visszaemlékezünk a dilatált konvolúció az, amelyik a bemeneti pixeleket nem szomszédos helyekről, hanem a dilatációs faktornak megfelelő számú pixelt átugorva szedi ki a képből. Tehát egy 3×3-as knvolúció 2-es dilatációval egy 5×5-ös ablaknak megfelelő helyről szedi a pixeleket, vagyis ekkora az effektív látómezeje. A dilatált konvolúciókból összerakott hálózat tehát egy lényegesen nagyobb részt tud belátni az egész képből a számítási kapacitás jelentős növelése nélkül.

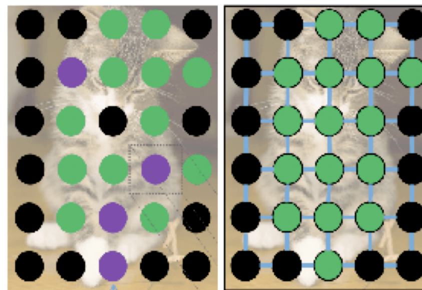
Ahogy a detektáló hálóknál, ugy itt is használják a piramis pooling trükkjét, azonban ezt is gyakran dilatált konvolúciókkal megoldva. Ebből keletkezik az Atrous Spatial Pyramid Pooling (ASPP) modul, amely szinte az összes modern szegmentáló architektúra szerves eleme.



7.7. ábra. *Az ASPP Modul.*

7.1.3. CRF

A tipikus FCN architektúrának van egy jelentős problémája: a pixelek ugyanis egymástól függetlenül kerülnek osztályozásra, így elképzelhető, hogy az egyes objektumokban furcsa lyukak, vagy szigetek keletkeznek téves osztályzások miatt. Ráadásul a leskálázás során a finom felbontású pozíció információnak jelentős része elveszik (különösképp, ha max poolingot használunk strided konvolúció helyett), aminek eredményeképp az objektumhatárok pozíciói pontatlanok lehetnek. Ennek elkerülésére alkalmazhatjuk a Conditional Random Field (CRF) nevű eljárást, amelynek alapötlete, hogy a végső osztályzást a szomszédos pixelek címkéi alapján egy konszenzus kialakításával oldja meg.



7.8. ábra. *A CRF alapelve.*

A CRF során a kép pixeleit egy gráfként képzeljük el, ahol az összes szomszédos pixel kapcsolatban áll. A gráfon két potenciálfüggvényt definiálunk: Az egyik a saját (Unary) potenciál, amely a pixelek osztály valószínűségétől fordított arányosság szerint függ (értsd: ha pixelnek a legvalószínűbb osztály címkét adjuk, akkor ez a potenciál minimális). Ezzel ellentétben a páros potenciál a környező pixelek és az adott pixel közötti interakciótól függ az alábbi módon:

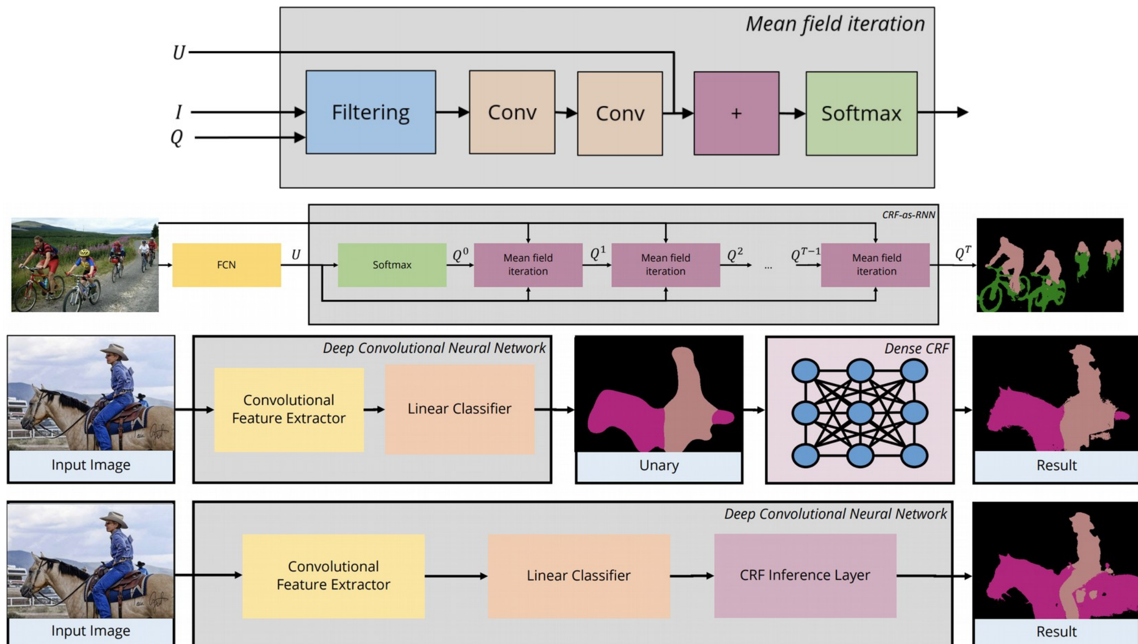
$$
\begin{gathered}
E = \sum_{i \in V} \Psi_i(x_i = l_i) + \sum_{i,j \in E} \Psi_{i,j}(x_i = l_i, x_j = l_j) \\
\Psi_{i,j} = \mu(l_i, l_j) \sum_m w^m \sum_{j \neq j} k_g^m(f_i, f_j) \\
k_g^m(f_i, f_j) = w_1^m e^{-\frac{||p_i - p_j||^2}{2\sigma^2}} + w_2^m e^{-\frac{||p_i - p_j||^2}{2\sigma_1^2} - \frac{||I_i - I_j||^2}{2\sigma_2^2}}
\end{gathered}
\tag{7.5}
$$

Ahol $\mu(l_i, l_j)$ az i-edik és j-edik címke közti kompatibilitás súlya, k_g a szomszédos pixelek hatását súlyozó RBF (Radial Basis Function - voltaképp egy gauss súly) kernel, w^m az m-edik hatás súlya, p_i és I_i pedig az i-edik pixel pozíciója és intenzitása. Ψ_i az -iedik pixel saját, $\Psi_{i,j}$ pedig az i-edik és j-edik pixelek közti páros potenciál. Érdemes megjegyezni, hogy a szomszédos pixelek relatív hatása alapvetően a központi pixeltől való távolságtól és a köztük lévő intenzitáskülönbségtől függ. Ez utóbbi akkor számít igazán, ha az adott pixel közel van. Ez a tag segíti a CRF algoritmust abban, hogy a szegmentációnál a címke váltások minél inkább a képi élekre simuljanak rá.

A CRF algoritmus lényege, hogy a két potenciáltagból adódó teljes potenciált a teljes képre minimalizálja. Ez egy NP-nehéz probléma, azonban a mean field iteráció algoritmusa a jelen esetben tud egy meglehetősen jó minőségű közelítést adni. Ennek során első lépésként a saját potenciálokat egyelővé tesszük a szegmentáló háló kimenetével. Ezt követően a saját potenciálokat a szoftmax függvénnyel skálázva előállítunk egy osztályozást. Ezt követően előállítjuk a páros potenciál ezen osztályozással súlyozott értékét, majd ezt levonjuk a saját potenciál értékéből. Ezután még egyszer alkalmazva a szoftmax függvényt előállítjuk az osztályozás új értékét:

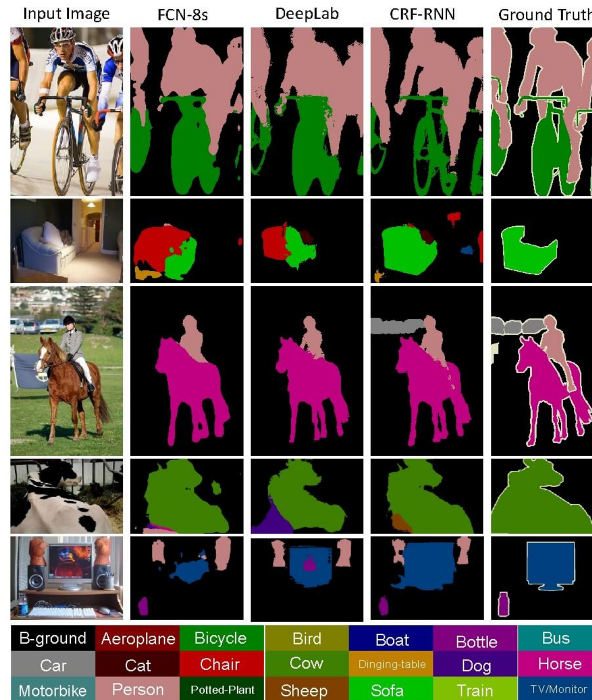
$$
\begin{aligned}
Q_i(l) &= \frac{1}{\sum_{l'} U_i(l')} e^{U_i(l)} && \leftarrow \quad \textit{Initialization} \\
\hat{Q}_i(l)^m &= \sum_{i \neq j} k^m(f_i, f_j) Q_j(l) && \leftarrow \quad \textit{Messages} \\
\hat{Q}_i(l) &= \sum_m w(m) \hat{Q}_i(l)^m && \leftarrow \quad \textit{Filter Weighting} \\
\hat{Q}_i(l) &= \sum_{l'} \mu(l, l') \hat{Q}_i(l') && \leftarrow \quad \textit{Compatibility} \\
\hat{Q}_i(l) &= U_i(l) - \hat{Q}_i(l) && \leftarrow \quad \textit{Add Unary Potential} \\
Q_i(l) &= \frac{1}{\sum_{l'} \hat{Q}_i(l')} e^{\hat{Q}_i(l)} && \leftarrow \quad \textit{Softmax}
\end{aligned}
\tag{7.6}
$$

Ahol $U_i(l)$ az i-edik pixel l-edik címke saját potenciál értéke, $Q_i(l)$ pedig ugyanezen pixel és címke osztályozása. A felső index több különböző közbenső szűrőt jelez, ennek fizikai jelentése nincs. Fontos megjegyezni, hogy az iteráció minden lépése leírható egy deriválható neurális háló szűrővel az alábbi módon:



7.9. ábra. *Egy Mean-Field iterációt elvégző struktúra, és az algoritmust elvégző neurális háló (felül). A CRF megoldása neurális hálóval (alul).*

Maga a teljes mean field iteráció módszere pedig ennek a cellának a visszacsatolt neurális hálóként történő alkalmazásával előállítható, a különböző súlyok pedig a teljes neurális hálóval együtt taníthatók.



7.10. ábra. *A CRF eredménye.*

7.1.4. Költségfüggvények

Fontos még szót ejteni a szegmentáció során használatos költségfüggvényekről és mérőszámokról. A szegmentáció pontosságára gyakori mérőszám a már korábban ismertetett Intersection over Union (IoU), csak itt nem téglalapok metszetét és unióját, hanem teljesen általános alakzatok esetében számoljuk ki ezeket. A szegmentálás során ezt a mérőszámot gyakran nevezzük Dice-koefficiens-nek is.

Mivel a szegmentálás alapvetően pixelek osztályozása, ezért az osztályozásnál megismert hibafüggvények minden további nélkül alkalmazhatók. Ennek megfelelően a mélytanulás korai szakaszában teljesen sztenderd volt a kereszt-entrópia pixelenkénti alkalmazása. Ennek viszont van egy szignifikáns hátránya, mégpedig az, hogy a szegmentálás erősen szenved a data imbalance problémájától, vagyis attól, hogy egyes osztályokból nem ugyanannyi van a tanító halmazban. Ez teljesen érthető, hiszen különböző osztályok általában eltérő méretűek, tehát nem ugyanannyi pixelből állnak.

Ez viszont ahhoz vezet, hogy nagy osztályok sokkal erősebben lesznek büntetve, mivel a CE hibafüggvény pixelenként számolódik, ergo a háló ugyanannyi büntetést kap egy nagy objektum határainak pontatlan becsléséért, mint egy kis objektum totális elvesztéséért. Ezt elkerülendő célszerű lenne egy olyan költségfüggvény, ami arányos az objektum méretével. Ebből könnyen adódna az eredeti Dice-együttható, azonban ez nem deriválható. Létezik azonban egy Soft-Dice nevű függvény, amely az eredetinek egy simított, deriválható változata:

$$L_{DICE} = 1 - \frac{2\sum y_p y_t}{\sum y_p^2 + \sum y_t^2}, \tag{7.7}$$

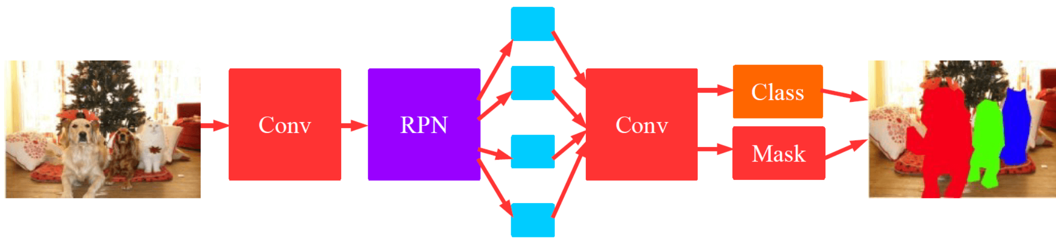
ahol y_p a háló kimenete egy adott pixelre, y_t pedig a ground truth értéke ugyanarra a pixelre. Fontos megjegyezni, hogy e fenti képlet bináris szegmentálás esetében értelmezhező, tehát y_p egy $[0, 1]$ közti valószínűség, míg y_t egy bináris érték. Természetesen, amennyiben több osztály közül szegmentálunk, akkor a hibát minden osztályra külön számoljuk, majd ezeket átlagoljuk.

7.2. Egyéb szegmentációs módszerek

A következő részben röviden szót ejtenénk a további szegmentációs módszerekről, vagyis az instance szegmentációról, ahol az azonos osztályhoz tartozó különböző objektumpéldányokat is megpróbáljuk elkülöníteni, illetve a szemantikus és instance szegmenáticók kombinációjának tekinthető panoptikus szegmentációról is.

7.2.1. Mask-R-CNN

Habár az instance szegmentáció érezhetően az egyik legnehezebb feladat az összes közül, az eddigi ismeretek alapján egy ilyen architektúra mégis könnyedén megérthető. A régió alapú objektumdetektálás (R-CNN és változatai) során az egyes objektumokat tartalmazó képrészleteket ugyanis már előállítottuk, így a feladatunk csak annyiban különbözik, hogy a befoglaló téglalap mellett minden objektumhoz egy bináris maszkot is elő kell állítanunk. Ezt könnyedén megtehetjük a szemantikus szegmentálásból ismert felskálázó hálórész segítségével. Ezt az architektúrát Maszk R-CNN néven ismerik.



7.11. ábra. *A Mask-R-CNN architektúra.*

Természetesen ezt a kiegészítést a YOLO architektúrákhoz is el lehet végezni, habár ott a felskálázó szegmentáló részt minden grid cellához külön elő kell állítani, így ez single-shot detektorok esetében valamivel nehézkesebb. Nem is meglepő, hogy a YOLO újabb változatai instance szegmentációt is tudnak végezni, így a detekció és szegmentáció közti határ kezd eltűnni.

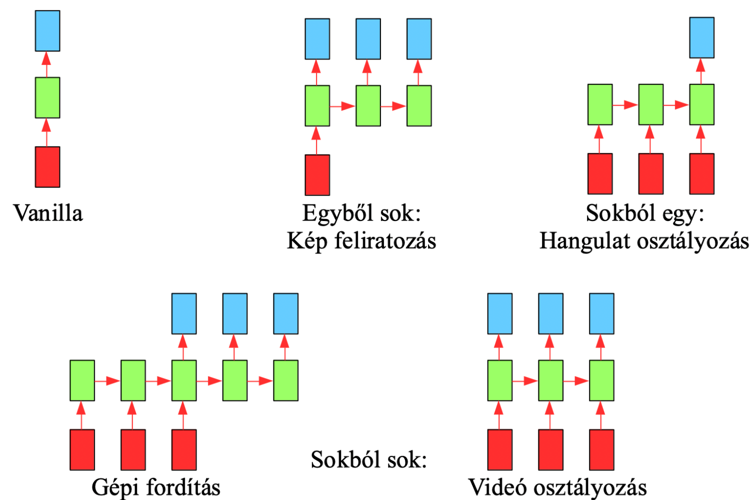
7.12. ábra. *A panoptikus szegmentáció eredménye.*

Végezetül az utolsó feladat előállítása még egyszerűbb, a panoptikus szegmentálás ugyanis egyszeren az előző két fajta kombinációja. Első lépésként az osztályainkat felosztjuk két kategóriára, melyek közül az első a ”things”, melyek olyan megszámlálható objektumok, melyek detektálhatók, így ezeken az instance szegmentáció elvégezhető. A második kategóriánk a ”stuff”, amelyben olyan

osztályok vannak, melynek számossága nem értelmezhető, vagy nem releváns (ilyenek például az úttest, ég, növényzet, épületek). Ezeken az osztályokon szemantikus szegmentálást végzünk, majd a két kapott szegmens képet elemenként szorozva kapunk egy végső szegmentációt.

7.3. Videók feldolgozása

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



7.13. ábra. *Különböző sorozatfeldolgozási feladatok.*

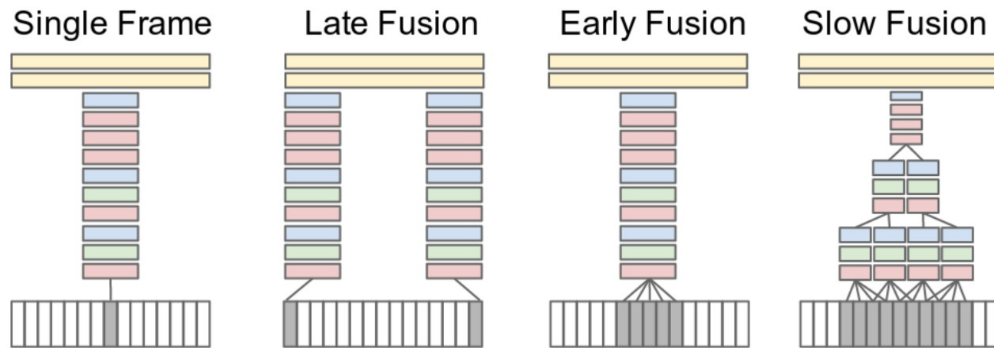
7.3.1. Multi-frame hálók

Érdemes megjegyezni, hogy amennyiben csak nagyon rövid időbeli összefüggéseket kell tudni kezelni, akkor lehetséges a háló bemenetére egyszerre két vagy több képet adni, vagy esetlegesen két kép különbségét felhasználni. Ennek legegyszerűbb módja a korai fúzió, amikor a képkockákat a háló elején, pixelszinten fűzzük össze. Ehhez a képkockákat a csatorna dimenzió mentén összefűzve kapunk egy $N \times C$ csatornaszámú bemenetet, amit egy hagyományos háló már fel tud dolgozni. Bár ez a legkevésbé számításigényes megoldás, hátránya azonban, hogy a pixel szinten történő összefűzés gyakran figyelmen kívül hagyja a kontextust és rossz eredményt adhat nagyobb mozgások esetén.

Másik végletként lefuttathatjuk ugyanazt a konvolúciós hálót két képkockán, majd csak a hálók végén végezzük el az összekapcsolást. Ebben az esetben már csak magas szintű szemantikus információt kell fuzionálnunk, ami sokkal könnyebben megtehető, azonban a hálót N-szer kell lefuttatnunk. Célszerű azonban a kettő között egy kompromisszumot elérni, amikor a képkockák közti információ összekapcsolását a hálón előrefelé haladva folyamatosan végezzük el. Ezt nevezzük lassú fúziónak.

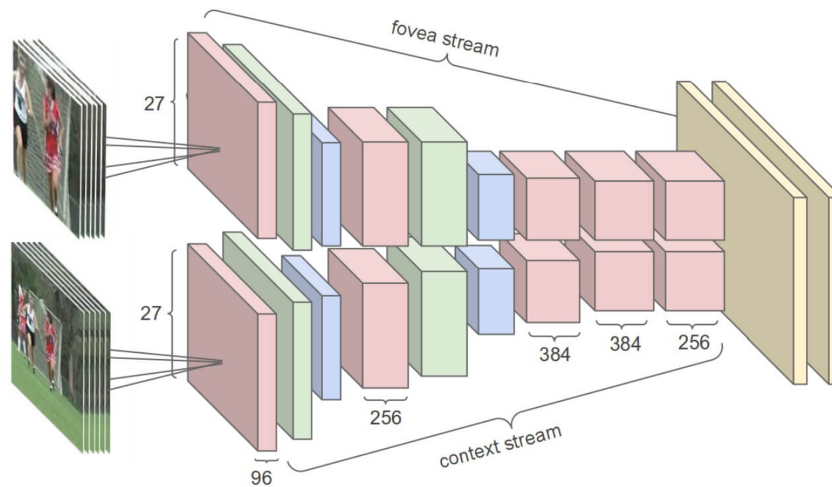
7.3.2. Többfelbontású hálók

Gyakori trükk még, hogy az emberi látáshoz hasonló architektúrát alkalmazunk. A szem középső, foveának nevezett része ugyanis lényegesen több csapot és pálcikát tartalmaz, mint a környező



7.14. ábra. *Az időben eltérő helyről érkező információk fúziós módszerei.*

részek. Léteznek olyan neurális hálók, amik két külön úttal rendelkeznek, melyek közül az egyik a fovea út, a kép középső részét dolgozza csupán fel. A másik út, a kontextus út a teljes képen végez feldolgozást, azonban mindezt alacsonyabb felbontáson. A két háló kombinálásának előnye a nagy felbontású feldolgozás megtartása a számítási igény kordában tartása mellett.

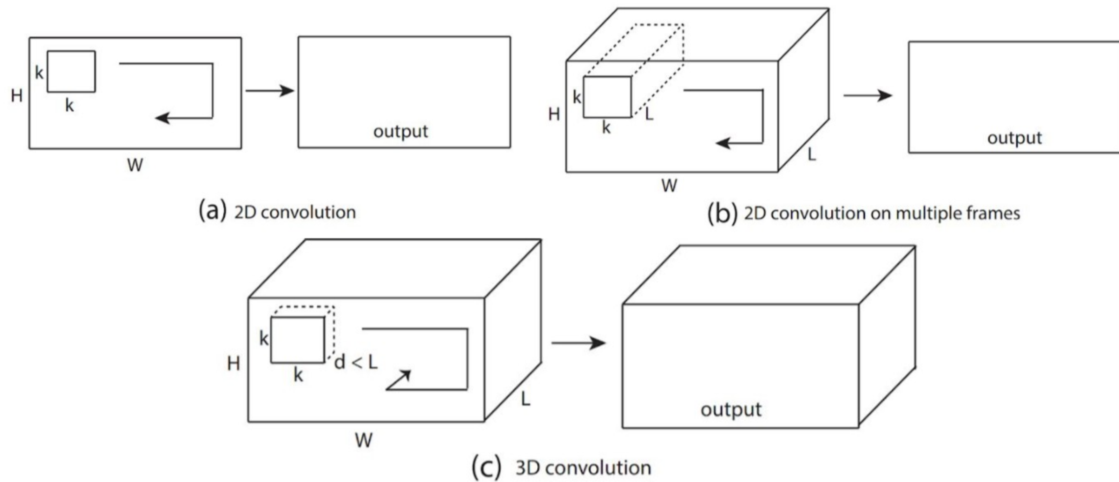


7.15. ábra. *A többfelbontású hálók tipikus architektúrája.*

7.3.3. 3D konvolúció

A korábban ismertetett módszernek van azonban egy hátránya: ezeknél a megoldásoknál az időbeliséget a csatorna dimenzió tárolja, amit a 2D konvolúciós rétegek redukálnak. Ennek következtében nehézkesen tudunk 2D konvolúciók segítségével időbeli jellemzőket megtanulni. A konvolúció ugyanis általánosítható N dimenzióba, így 3D konvolúciók használatával könnyedén létrehozhatunk olyan architektúrákat, amik a különböző tér és időbeli jellemzőket képesek külön kezelni. Ez a megoldás lehetővé teszi továbbá az idő dimenzió menti paraméter megosztást, ami jelentős paraméterszám csökkenéssel jár.

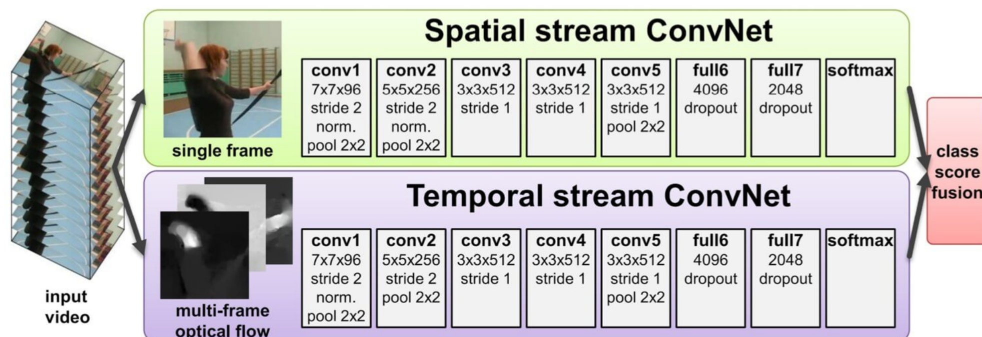
Fontos megjegyezni, hogy a korábban említett információ fúziós paradigmák közül a 3D konvolúciók a slow fusion egy természetes megoldásának tekinthetők, hiszen az egymás után csatolt véges méretű konvolúciók egyre növekvő látómezővel rendelkeznek. Mivel a 3D konvolúció az idő dimenzióban is konvolvál, ezért az egymást követő rétegek egyre nagyobb időtartamot látnak be.



7.16. ábra. *A 2D (felül) és a 3D (alul) konvolúció működése.*

7.3.4. Multimodális hálók

A legutolsó alapvető megoldás az úgynevezett multimodális bemenetű hálók alkalmazása. Ezek a hálók több, jelentősen eltérő jellegű bemenettel rendelkeznek. Ezek közül az egyik jellemzően az aktuális képkocka, a másik pedig egy valamilyen időbeliséget leíró bemenet. Ez a második bemenet lehet például különbségkép, vagy éppen egy optikai áramlás kép. Az ilyen architektúrák esetén komoly kihívást jelent a két modalitás pontos illesztése, ugyanis bizonyos esetekben ezek el lehetnek csúszva egymáshoz képest. Érdemes megjegyezni, hogy multimodális hálókat alkalmaznak különböző érzékszervi bemenetek illesztésére is (pl.: kép és hang).



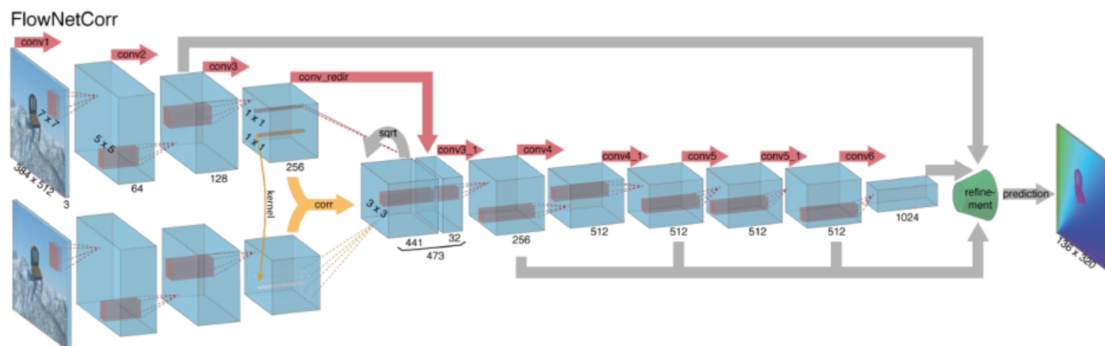
7.17. ábra. *Egy multimodális háló tipikus felépítése.*

A korábbi bekezdésben említettem az ún. optikai áramlást, arról viszont nem esett szó, hogy ez pontosan mi is. Az optikai áramlás esetében azt feltételezzük, hogy két egymást követő időpillanatban készített kép áll rendelkezésünkre, és az ezek között történt mozgást szeretnénk leírni. Ezt megtehetjük úgy, hogy a kép minden pixelének megkeressük a párját a másik képen (vagyis azt a pontot, ahova a két kép között az eredeti pixel elmozdult), és a kettőt összekötő elmozdulás vektort hozzárendeljük az egyes pixelekhez. Az így keletkezett képet egy tipikus áramlásképre hasonlít, azonban az áramló "közeg" nem valamilyen folyadék, hanem maguk a pixel intenzitások.

Az optikai áramlás meghatározására léteznek hagyományos módszerek, melyek az elmozdulást lokális környezetek vizsgálata alapján keresik, ezen módszerek ismertetése azonban nem része a jelen tárgynak. Természetesen a hagyományos módszeren túl lehetőség van arra is, hogy az optikai áramlást két bemeneti kép felhasználásával egy konvolúciós neurális háló segítségével becsüljük. Érdemes azt megjegyezni, hogy az optikai áramlás meghatározására létező hagyományos algoritmusok teljesítménye meglehetősen jó (főleg, mivel az optikai áramlás kifejezetten alacsony szintű látási feladat), így a neurális hálók alkalmazási lehetőség inkább tartozik az érdekesség kategóriájába.

7.18. ábra. *Az optikai áramlás.*

Az egyik legyakrabban alkalmazott háló architektúra az úgynevezett FlowNet, amely három fő részből áll. Először a két bemeneti képen külön-külön lefuttat egy hagyományos leskálázó jellegű konvolúciós hálórészt (mindkét képre ugyanazokkal a súlyokkal). Ezt követően a két képből kapott aktivációs tömböt egy korrelációs réteg segítségével egyesíti, majd az így kapott tömbön egy újabb leskálázó hálórészt futtat le. A háló utolsó része egy felskálázó jellegű réteg, ami az aktivációs tömb térbeli dimenzióit minden lépésben növeli, hogy az eredeti képpel megegyező méretben állíthassuk elő a kimenetet.



7.19. ábra. *A FlowNet architektúra.*

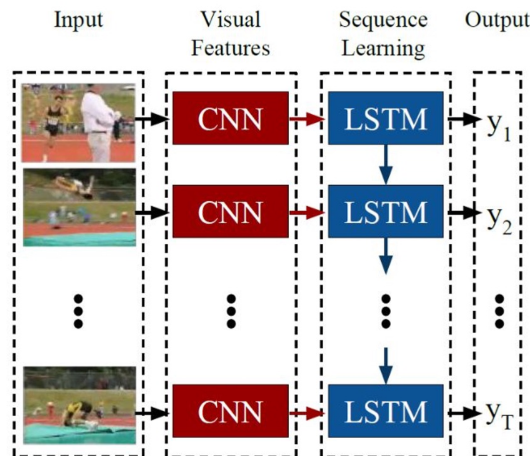
A FlowNet egyik fontos eleme a két képből származó információ összeolvasztását végző korrelációs réteg. Amennyiben adott két bemeneti aktivációs tömb f_1 és f_2, akkor a korrelációs réteg az első tömb egy adott x_1 és a második tömb x_2 pozíciói körüli kxk környezetek közti korrelációt számolja ki az alábbi módon:

$$C(x_1, x_2) = \sum_{p=(-k,-k)}^{(k,k)} f_1(x_1 + p)^T f_2(x_2 + p) \tag{7.8}$$

Ezt a két tömb összes lehetséges pozíció párjain végigfuttatva a kapott aktivációs tömb térbeli dimenziója a két eredeti tömb méretének szorzata lenne, ami érezhetően rendkívül nagy méretet jelentene. Mivel azonban a két képkocka közötti elmozdulás várhatóan relatíve kicsi, ezért elég a korrelációt egy adott x_1 pozíció környezetébe tartozó x_2 pozíciókban kiértékelni. Érdemes visszaemlékezni, hogy a korreláció művelet eredménye azoknál a pozíció pároknál lesz nagy, ahol a két tömb aktivációi erős statisztikai összefüggést (hasonlóságot) mutatnak.

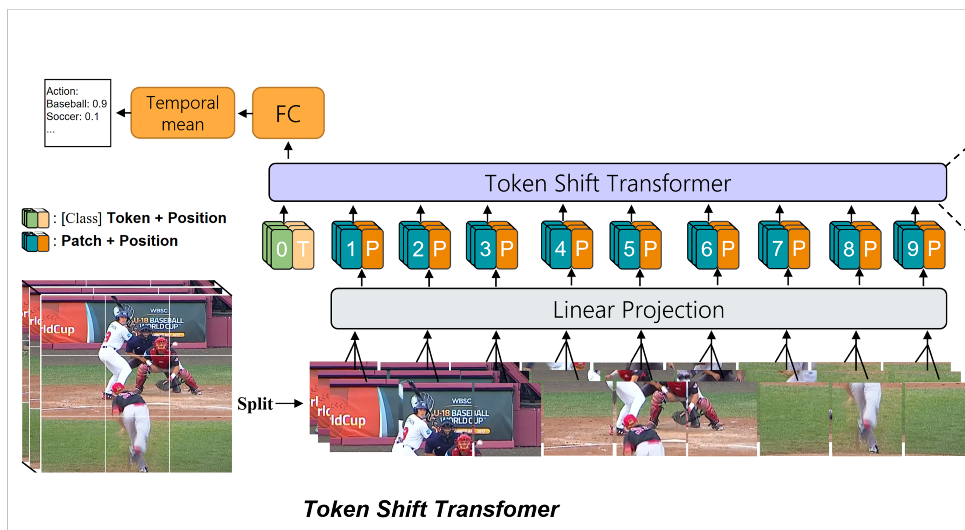
7.4. Videó osztályozás

A fejezet végén fontos még említést tenni a különböző videó osztályozó architektúra megoldásokról. Habár ez az eddigiek alapján kitalálható, a videók osztályozásának egyik legkézenfekvőbb módja, ha egy konvolóciós backbone hálózat végére egy visszacsatolt (célszerűen LSTM) réteget csatolunk, ezáltal megvalósítva egy memóriával rendelkező modellt. Megjegyzendő, hogy ez egy late fusion paradigma, hiszen az időbeli kombináció a háló végén, magas szinten történik, ugyanis az LSTM alacsony szinten történő beépítése konvergencia szempontjából problémás.



7.20. ábra. *Videó osztályozás LSTM-ek segítségével.*

Természetesen ahogy azt a korábbi fejezetekben említettük, szekvenciafeldolgozás terén a transzformereknek komoly előnye van az RNN-ekkel szemben, ugyanis numerikus problémák nem lépnek fel ezekkel. Így elvileg jobb megoldást jelenthet, ha a konvolúciós backbone végére LSTM helyett egy időbeli transzformert teszünk. Természetesen, ha ezt online módon (vagyis nem egy teljes videóra utólag, hanem az egyes frame-ek érkezésekor valós időben) szeretnénk alkalmazni, akkor szükséges a transzformer folyamatos újbóli alkalmazása, eggyel több adattal. Ez azonban problémás, hiszen akkor a szekvencia hossza és a szükséges memória és számításigény folyamatosan nő.



7.21. ábra. *Videó osztályozás transzformerek segítségével.*

Ez a problémát orvosolja a TokenShift Transzformer, amely egy alacsony költségű operáció segítségével a legrégebbi időbeli pillanathoz tartozó tokeneket kishifteljük a transzformerből, az újakat

pedig be. Így megoldható a változó időbeli információ benntartása anélkül, hogy folyamatosan mindent újra kelljen számolni.

8. fejezet

3D Feldolgozás

8.1. Bevezetés

A korábbi fejezetek során részletesen tárgyaltuk a képsorozatok feldolgozását, valamint a detektálás és szegmentálás feladatát. A jelenlegi előadás célja, hogy megtárgyaljuk, hogy milyen módon lehetséges ugyanezeket a feladatokat 3D struktúrák esetében elvégezni.

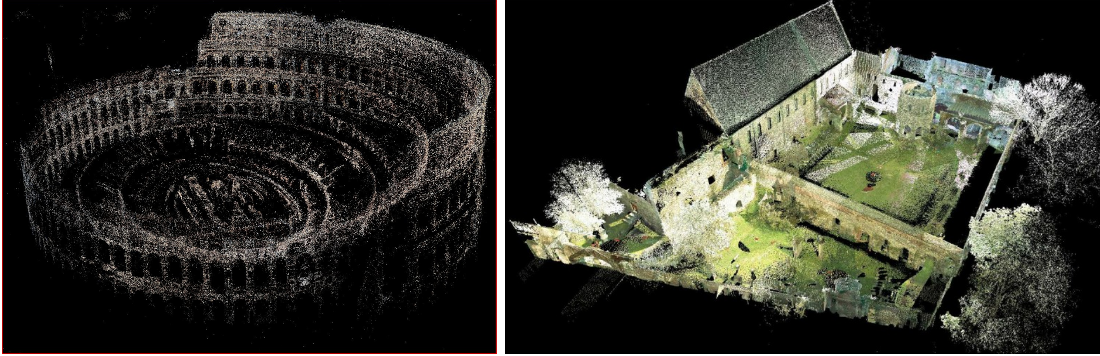
8.2. 3D Struktúra reprezentációja

Mielőtt azonban belekezdnénk az egyes módszerek ismertetésébe, egy alapvető kérdést szükséges még tisztáznunk. Az ugyanis nem nyilvánvaló, hogy a térbeli struktúrát milyen formában reprezentáljuk. A hagyományos számítógépes látás egy kétdimenziós képet egy kétdimenziós rácson tárol, ahol a rács minden eleme egy pixel értéke. Magától értetődő megoldás volna egy térbeli struktúrát úgynevezett voxelek (a volumetrikus és a pixel szó összemosása) háromdimenziós rácsaként tárolni, ahol minden voxelhez egy kicsi kocka alakú térfogatrész tartozik, a voxel értéke pedig a hozzá tartozó térrészben található objektumtól függ.

Az előbb ismertetett megoldás azonban egy meglehetősen rossz ötlet több szempontból is, melyek közül az első a dimenzionalitás átka. Képzeljük el, hogy egy áltagos felbontású kép mindkét dimenziójában 1-4000 pixelt tartalmaz, azaz összesen néhány millió pixelből áll, vagyis a memóriában néhány megabájt méretet foglalnak. A térbeli struktúráknak azonban eggyel több dimenziója van, tehát hasonló felbontás mellett a voxelek száma néhány milliárd lesz, vagyis a szükséges memória a gigabájtos tartományban mozogna. Arról nem is beszélve, hogy a rekonstrukciót gyakran számos képből állítjuk elő, így míg egyetlen kép a teljes térnek csak egy kis részét látja, a rekonstrukción az egész szerepelni fog, ahhoz pedig a képeknél megszokottnál nagyobb felbontásra volna szükségünk.

További probléma a voxelrácsos megoldással, hogy rendkívül pazarló. Míg általában egy kép minden pixelére esik fény, szemben a háromdimenziós terek meglehetősen üresek. Bíztatom az olvasót, hogy nézzen körül; amennyiben nem egy raktárhelységben tartózkodik, akkor a helység teljes térfogatának a 90-95 százaléka valószínűleg üres. Ehhez vegyük hozzá, hogy természetüknél fogva a háromdimenziós rekonstrukciók csak az objektumok felületét tartalmazzák (a kamerák nem látnak be az objektumok belsejébe), így egy átlagos rekonstrukció térfogatának nagyjából 1 százalékában található bármi is.

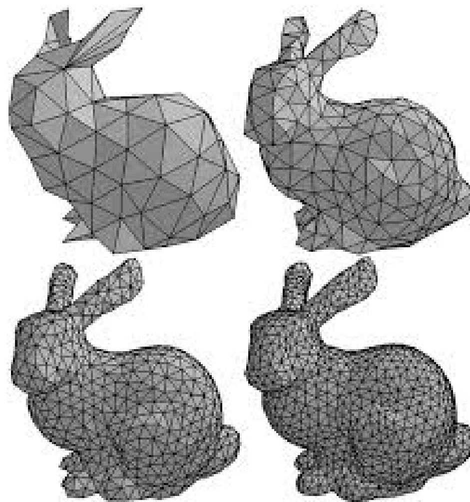
Éppen ezért a háromdimenziós struktúrák alapvető tároló struktúrája a pontfelhő, amely egyszerűen az objektumokat alkotó háromdimenziós pontok (általában rendezetlen) listája. A fent tárgyaltak fényében ez a módszer takarékosabb és pontosabb is, mint a rács használata. Érdemes megjegyezni, hogy a pontfelhőkben lévő pontok a pozíciójukon felül egyéb adatokat is gyakran tárolnak a gyakorlatban. Ezek közül az egyik legyakoribb eset, amikor minden ponthoz hozzárendeljük annak a pixelnek a színét, amely az egyes képeken az adott ponthoz tartozik. Szintén



8.1. ábra. *Bináris (bal) és színes (jobb) pontfelhők.*

gyakori, hogy a pontok alkotta felületek normális vektorát kiszámítjuk, és minden ponthoz hozzárendeljük.

Érdemes még megemlíteni, hogy a pontfelhőn kívül léteznek még magasabb szintű leírási módszerek háromdimenziós objektumok esetére. Különböző struktúrákat lehetséges különböző primitív formákkal (gömb, sík, henger, kúp stb.), vagy paraméteres felületekkel és görbékkel közelíteni, így potenciálisan nagy ponthalmazokat csupán néhány számmal leírni. Szintén elterjedt megoldás a háromszögháló (mesh) használata, ahol a komplex felületeket háromszög alakú építőelemekkel közelítenek. Ez utóbbi módszer elsősorban a számítógépes grafikában elterjedt.



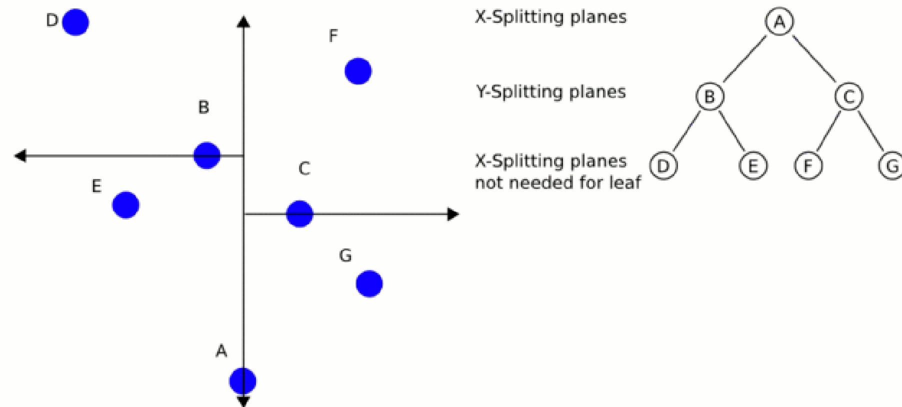
8.2. ábra. *Egy mesh segítségével reprezentált felület eltérő részletességgel.*

8.2.1. Kd-fa reprezentáció

Érdemes megjegyezni, hogy pontfelhőkben gyakorta szükségünk van az egyes pontok közeli, vagy legközelebbi szomszédjainak megkeresésére. Képek esetén ez egyszerű, hiszen egy pixel legközelebbi szomszédjai ott vannak mellette. Pontfelhők esetében azonban a pontok nincsenek feltétlenül bármilyen logikus sorrendben, így a legközelebbi szomszéd megtalálásához N darab pontot végig kell ellenőriznünk, ahol N a pontfelhőben lévő pontok száma.

Ezt a problémát próbálja orvosolni az úgynevezett kd-Fa struktúra. A kd-fa a pontfelhő pontjait egy gráfban helyezi el, ahol egy gyökér csomópont található, és minden csomópontnak két gyereke van (más szóval a kd-fa egy bináris fa). A fa konstruálásának kezdetekor kiválasztunk egy pontot gyökérpontnak, majd egy tetszőleges dimenzió mentén egy síkkal kettéválasztjuk a pontfelhőt (ennek a síknak tartalmaznia kell a pontot). Ezt követően megkeressük a síkhoz legközelebb eső

pontot mind a két részpontfelhőből, és ezek a pontok lesznek a kezdeti pont két gyereke. Ezt követően a két gyerekpontra megismételjük a gyökérpontra elvégzett műveleteket (pontot tartalmazó síkkal elmetszés, új gyerekpontok keresése) ügyelve arra, hogy a gyerekeknél használt metsző sík mindig merőleges legyen a szülőnél használt síkra.



8.3. ábra. *A kd-fa konstruálási szabálya.*

A kd-fa használatának hatalmas előnye, hogy a megkonstruálása után (amit csak egyszer kell megcsinálni), ha legközelebbi szomszédokat kell keresni egy pontfelhőben, akkor a bináris fa struktúrájából adódóan minden lépésben a pontfelhő egyik felét teljesen ki tudjuk zárni a keresésből. Ez azt jelenti, hogy N lépés helyett annak kettes alapú logaritmusával lesz arányos a keresési idő, ami hatalmas gyorsulást jelent, különösképpen nagy, többmillió pontot tartalmazó felhők esetében.

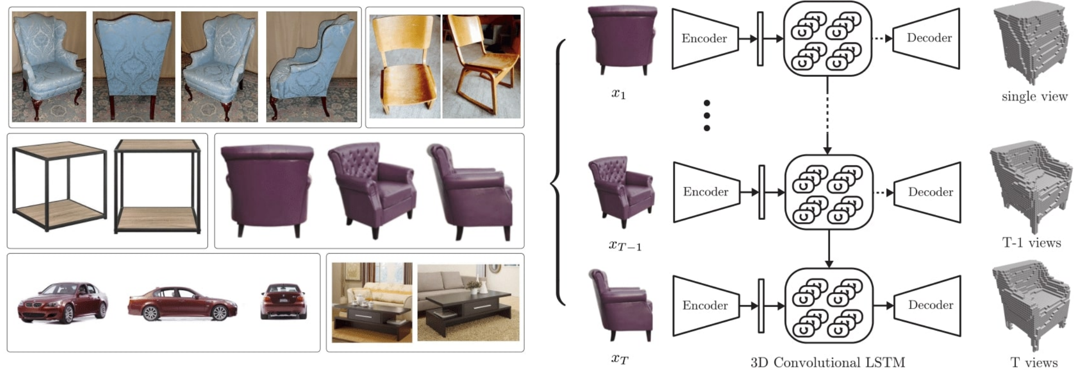
8.3. 3D információ előállítása

A háromdimenziós adatok feldolgozása előtt azonban érdemes körbejárnunk az a kérdést, hogy hogyan lehet ilyen jellegű információt előállítani. Természetesen használhatunk erre különböző 3D szenzorokat (sztereó vagy mélység kamera, LIDAR), azonban a 3D szenzorok általában egy nagyságrenddel drágábbak, mint egy hasonló képességű 2D kamera. Éppen ezért bizonyos esetekben célszerű lehet a 3D információt 2D képek felhasználásával visszaállítani, vagyis 3D rekonstrukciót végezni.

A 3D rekonstrukció elvégzésére léteznek hagyományos módszerek, a jelen tárgy témája azonban a mély neurális háló alapú rekonstrukció megvalósítása. Mindkét esetben érdemes azonban belátni, hogy a 2D képalkotás során az adott jelenet térbeliségére vonatkozó információk egy része elveszik, melynek következtében az eredeti 3D elrendezés visszaállításához valamilyen extra információra kell szert tennünk. A legtöbb esetben ezt az extra információt további, ugyanarról a jelenetről készített 2D képek szolgáltatják. Amennyiben összesen 2 képet használunk, akkor sztereó, több kép esetében pedig többnézetű rekonstrukcióról beszélhetünk.

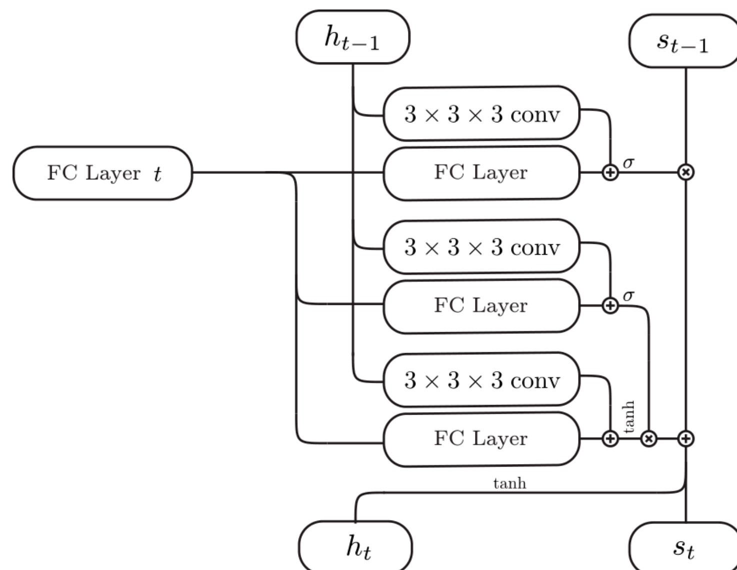
8.3.1. Többnézetű rekonstrukció

A többnézetű 3D rekonstrukció megvalósítására az egyik legelterjedtebb architektúra az úgynevezett 3D-RRNN (3D Reconstruction Recurrent Neural Network), melynek sablonos felépítése korábbi architektúrákból ismerős lehet. A háló alapvetően a szemantikus szegmentáló hálózatokhoz hasonlóan egy le- és egy felskálázó részből áll, annyi különbséggel, hogy a felskálázó rész nem 2D, hanem 3D konvolúciós rétegeket tartalmaz, így az általa előállított aktivációs tömb is háromdimenziós lesz. Az architektúra két fő részét egy LSTM réteg köti össze: Ez a réteg felel azért, hogy a háló bemenetére egymás után adott 2D képekből kinyert információt egyetlen 3D reprezentációba integrálja.



8.4. ábra. *A 3D-RRNN architektúra.*

Ez az LSTM cella azonban lényegesen különbözik a korábban tárgyalt hagyományos LSTM-től, ugyanis itt a belső állapot és a kimenet is egy 3D tömb szemben a hagyományos LSTM 1D vektorával. Ennek következtében az LSTM cella azon műveletei, amik a belső állapoton, vagy az előző kimeneti értéken hajtódnak végre szintén egy 3D konvolúció segítségével fogalmazhatók meg. Érdemes azt is megjegyezni, hogy az LSTM input gate működésének itt külön szemléletes szerepe van (emlékezzünk ez az a változó, amely eldönti, hogy a belső állapot melyik részébe írjuk bele az új változókat). Ebben az esetben az input gate feladata, hogy az aktuális kép nézőpontja alapján kiválassza a 3D belső állapot azon részét, amelyeket az adott nézőpontból látni lehet.



8.5. ábra. *A 3D LSTM cella felépítése.*

8.3.2. Egynézetű mélységbecslés

Érdemes megjegyezni, hogy habár a térlátáshoz legalább két szem/kamera szükséges, az emberi agy mégis képes egyes jelenetek térbeliségét csupán egyetlen kép alapján megbecsülni. Ehhez alapvetően két különböző információt használunk fel: egyrészt az életünk során rengeteg információt gyűjtünk bizonyos objektumok térbeliségéről, így azt egy egyszerű 2D kép alapján is meg tudjuk becsülni. Másrészt a jelenetben látható árnyékok és visszaverődések segítségével az egyes felületek irányát is megbecsülhetjük, ami alapján a 3D struktúra kitalálható.

Hasonló működést kíván elérni a DethNet, ami egy egynézetű mélységbecslő neurális háló. A háló feladata, hogy a bemenetére adott egy darab állókép alapján minél pontosabban meg tudja becsülni az ahhoz tartozó mélységképet (vagyis a kamerától való távolságot). A hálózat hibafüggvénye, hogy

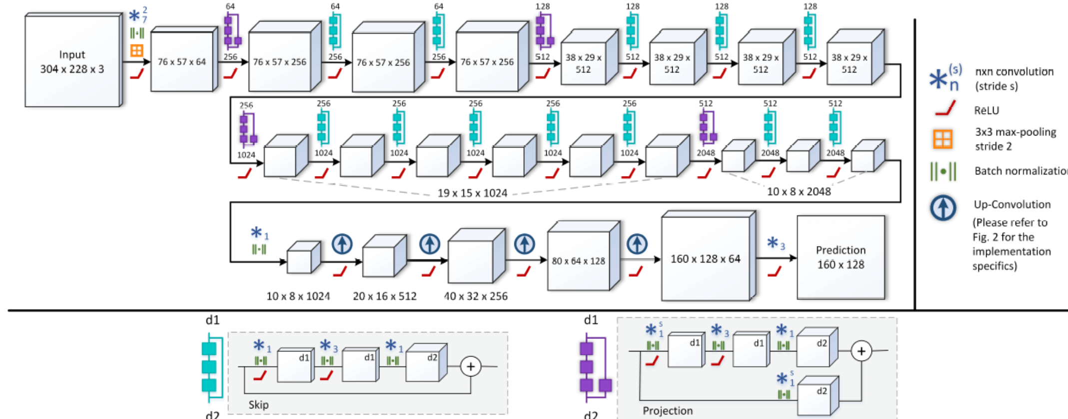
a célként megadott mélység értékeket minél nagyobb pontossággal eltalálja, vagyis célszerű lehet a hálót pixelenkénti négyzetes hibával büntetni.

Ezzel a módszerrel azonban az a probléma, hogy egy kép alapján nincs jó módszerünk arra, hogy a mélység értékeket abszolút értelemben pontosan megbecsüljük, ehelyett inkább csak az egyes pixelek relatív mélységét szeretnénk eltalálni. Ennek a működésnek az eléréséhez minden kép esetében a cél depth értékeket a $[0, 1]$ intervallumra skálázzuk, a hálót pedig az alábbi skálainvariáns költségfüggvénnyel büntetjük:

$$L = \frac{1}{N}\sum_i (log(\hat{d}_i) - log(d_i))^2 - \frac{1}{N^2}\Big(\sum_i log(\hat{d}_i) - log(d_i)\Big)^2 \tag{8.1}$$

Ahol $\hat{d}_i$ a háló által becsült mélység érték az i-edik pixelhez, d_i pedig az igazi mélység, N pedig a pixelek száma. Látható, hogy a költségfüggvény első tagja egyszerűen a becsült és az igazi mélységek logaritmusai közti négyzetes hiba. A második tag szerepe, hogy a költségfüggvényt jutalmazza abban az esetben, ha a becsült és igazi értékek közti eltérés a legtöbb pixel esetén ugyanabba az irányba történik. Ez akkor lesz igaz, ha a becsült és a referencia mélység képek között abszolút eltérés van, de a pixelek relatív mélységei helyesek a becslésben is.

A DepthNet architektúrája alapvetően a szemantikus szegmentáló hálózatokra emlékeztet, ugyanis egy közvetlenül kapcsolódó le- és felskálázó részből áll. A DepthNet fontos része, hogy a leskálázó rész reziduális blokkokból áll, a konkrét dimenziócsökkentést pedig az erre hasonlító projekciós blokkok végzik. Ezek a blokkok a térbeli dimenzió csökkenése miatt a bemenetet nem közvetlenül, hanem egy 1x1-es konvolúciós szűrőn keresztül adják a kimenethez. A felskálázást a háló szintén reziduális jellegű blokkok segítségével végzi.

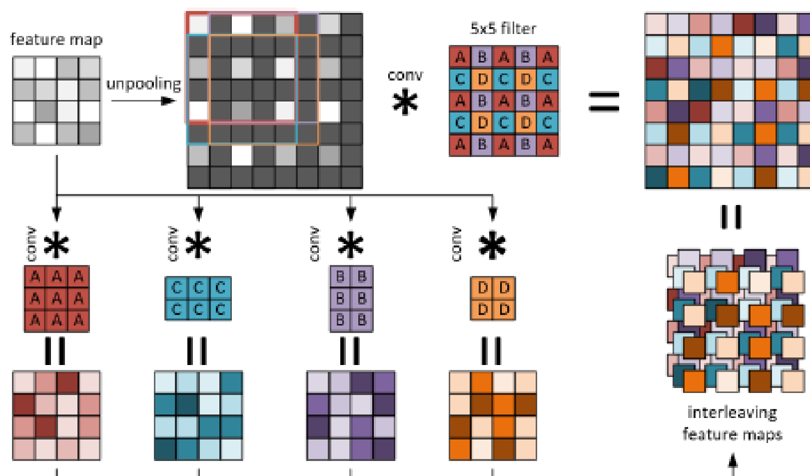


8.6. ábra. *A DepthNet architektúra.*

A felskálázás gyors megoldásához a DepthNet egy speciális megoldást használ. Az unpooling alapú felskálázás ugyanis általában a dimziónövelés után egy aránylag nagy (mondjuk 5x5) méretű konvolúciós szűrést is véget az aktivációs tömb simítása céljából. Ennek a műveletigénye $25 \times 4 \times N^2$. A DepthNet felskálázó szűrője azonban a helyett az eredeti méret mellett végez négy párhuzamos szűrést, eltérő méretű szűrőablakokkal, majd az így kapott négy aktivációs tömböket váltakozó módon összefűzi, így adva egy felskálázott tömböt. Az alábbi ábrán látható konvolúciós szűrők használata esetén a szűrés költsége $25 \times N^2$, vagyis negyede az eredetinek, azonos paraméterszám mellett.

8.4. 3D adatok feldolgozása

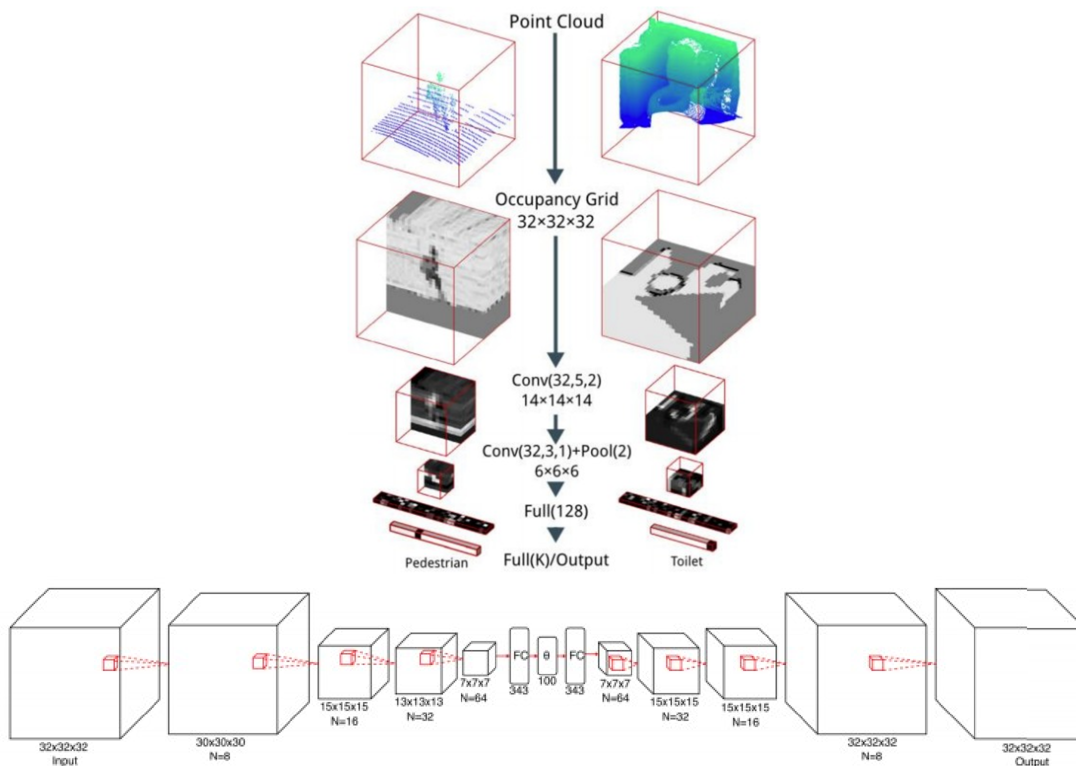
A deep learning módszereinek alkalmazása alapvető nehézségekbe ütközik a 3D feldolgozás esetén, ami jelentősen megnehezíti ezen módszerek használatát.



8.7. ábra. *A gyors felskálázó konvolúció elve.*

8.4.1. Voxel hálók

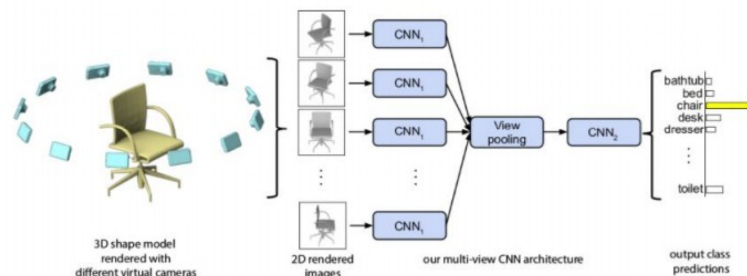
Adja ugyanis magát az ötlet, hogy használjunk 2D konvolúciós hálók helyett 3D hálót voxeles adatokon. Azonban ahogy azt az előadás elején megbeszéltünk, ez rendkívül problémás, mivel a tárolásnak nagy memóriaigénye van, ráadásul pazarló is. Ez a deep learning esetében különösen nagy probléma, mivel a mély neurális hálók már képek esetén is hatalmas memóriaigénnyel rendelkeznek, ami az egyik legfőbb szűk keresztmetszetet jelenti a modern GPU-k alkalmazásánál. Ennek ellenére léteznek voxel neurális hálók, ezek azonban elég kicsi felbontáson működnek csak, így a teljesítményük is limitált.



8.8. ábra. *Egy voxel alapú osztályozó (felül) és egy szegmentáló (alul) neurális háló.*

8.4.2. Projekción alapuló hálók

Léteznek még neurális hálók, amik a 3D pontfelhőből véletlenszerűen generált nézőpontokból 2D vetületeket állítanak elő a vetítés egyenlete alapján, majd ezt egy hagyományos 2D konvolúciós háló segítségével osztályozzák. Ezzel visszavezethető a 3D osztályozás művelete 2D osztályozásra, azonban ez is jelentős lassulással járul, hiszen több képet kell végigfuttatni ugyanazon a hálón. Továbbá ezen módszerek igen nehezen terjeszthetők ki további problémák (detektálás, szegmentálás) megoldására.



8.9. ábra. *A Projekció alapú mély tanulás sémája.*

8.4.3. Pontfelhő hálók

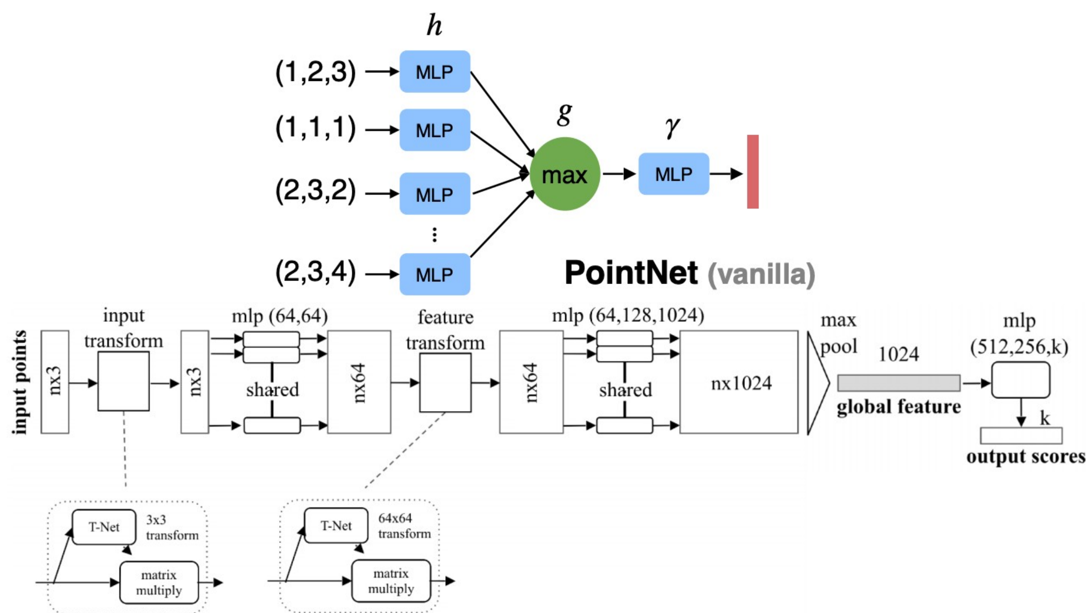
Ha viszont megkíséreljük valamelyik kézenfekvőbb reprezentációs módszert alkalmazni, akkor hamar szembesülhetünk azzal, hogy ezek kevésbé foghatók meg jól konvolúciós hálók segítségével. Pontfelhők esetében például azzal a problémával szembesülünk, hogy a neurális hálók alapvetően rendezett adathalmazokon tudnak jól működni. Ha a pontfelhő felsorolásában megcserélek két pontot, akkor még mindig ugyanazt a pontfelhőt írják le, egy hagyományos neurális háló viszont nem ugyanazt fogja végrehajtani.

Erre egy lehetséges megoldás, ha a neurális hálót szimmetrikus (kommutatív) függvényekből építjük fel, hiszen így a pontok sorrendje mindegy. Sőt, ha a pontokon egyesével elvégzünk különböző nem szimmetrikus transzformációkat, majd ezeknek vesszük egy szimmetrikus függvényét, akkor az eredő függvény is szimmetrikus lesz. Ezt az ötletet dolgozza ki a PointNet, melynek egy cellája a pontfelhő pontjait egyesével egy többrétegű hálón küldi végi, majd ezeknek veszi a maximumát. Ezen felül különböző tanulható transzformációkat (mátrixszorzás) is tartalmaz a háló, amelyek az egyes jellemzők közti interakciókat valósítják meg.

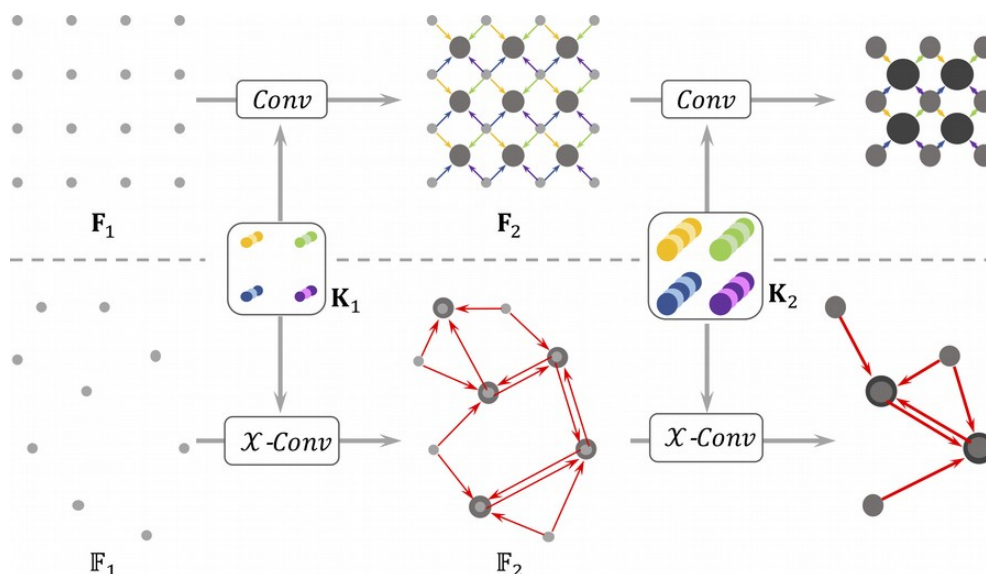
Egy másik lehetséges megoldás az, ha egy transzformáció segítségével a pontokat négyzetrácsos elrendezésbe hozzuk, majd ezt követően végezzük el a konvolúciót. Ez úgy képzelhető el, hogy minden pontnak megkeressük az egyes irányokba található legközelebbi szomszédját, majd ezeket tekintjük közvetlen szomszédnak a rácson. Az így elvégzett konvolúciót χ-Konvolúciónak nevezzük. A konvolúciót itt tovább módosíthatjuk úgy, hogy ne csak a szomszédos pontok/jellemzők értékeit, hanem azok távolságát is figyelembe vegye. Ezt a megoldást alkalmazza az úgynevezett Point-CNN architektúra.

8.4.4. Kd-fa hálók

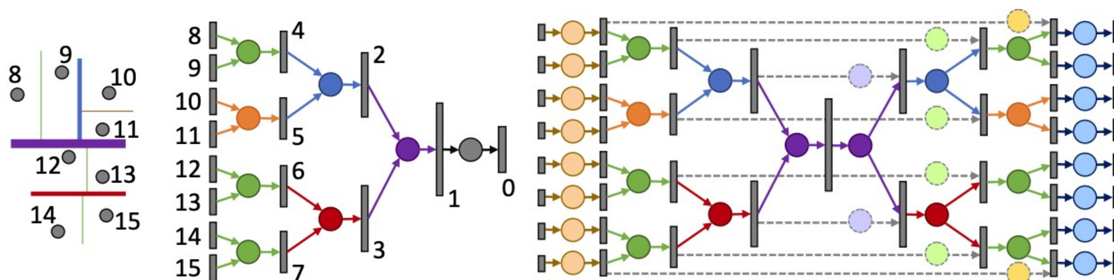
Egy említésre méltó megoldás még a kd-Net, amely a pontfelhő kd-fa reprezentációja alapján konstruál hálót. Ebben a reprezentációban a kd-fa szintjei felelnek meg a háló rétegeinek, így minden réteg két pont/jellemzőt fog egyre redukálni. Minden egyes rétegben az azonos irányú vágás által szeparált pontpárok paraméterei megegyeznek. Ez a háló architektúra használható osztályzásra, valamint szegmentálásra is.



8.10. ábra. *A PointNet egyetlen transzformációs rétege (felül) és a teljes struktúra (alul).*



8.11. ábra. *A hagyományos konvolúció (felül) és a χ-konvolúció (alul).*



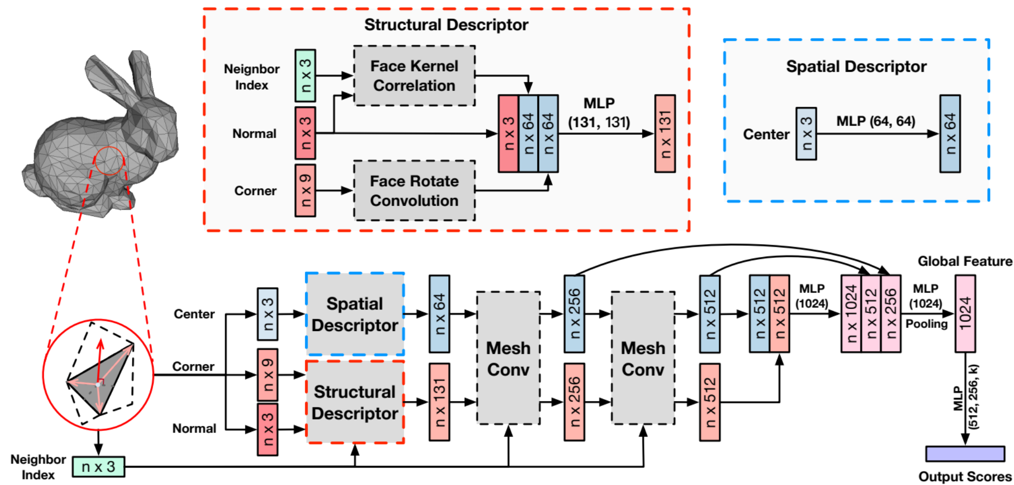
8.12. ábra. *A minta hálóhoz (bal) konstruált osztályozó (közép) és szegmentáló (jobb) kd-Net.*

8.4.5. MeshNet

Az egyik legújabb struktúra az ún. MeshNet, amely - nevéhez híven - mesh formában emgadott 3D struktúrákat vár bemenetként. A modell a mesh lapjainak paramétereiből (középpont, sarkok,

normális) térbeli és strukturális leírókat generál, majd ezeken egy mesh-konvolúciós hálózatot futtat le. A PointNet-hez hasonló módon a konvolúció végrehajtásához szükséges a szomszédosság definiálása, ez viszont a mesh esetében triviális: azok a háromszögek, melyeknek van közös oldala szomszédosnak számítanak.

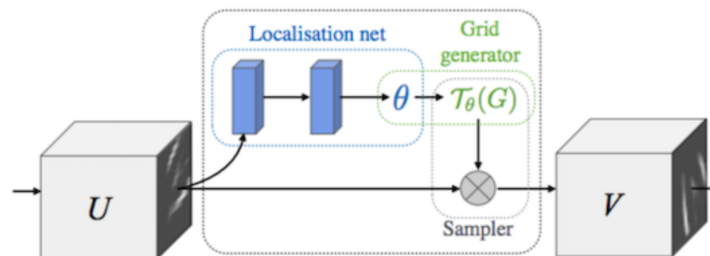
Érdemes még megjegyezni, hogy létezik a PyTorch3D könyvtár, amely a Facebook által készített hivatalos kiegészítő. Ez a könyvtár támogatja a mesh alapú neurális háló architektúrák tanítását, első sorban az alapvetően heterogén struktúrájú meshek manipulálásának megoldásával. Ezen felül a könyvtár részét képzi egy deriválható mesh rederelő is.



8.13. ábra. *A MeshNet architektúra.*

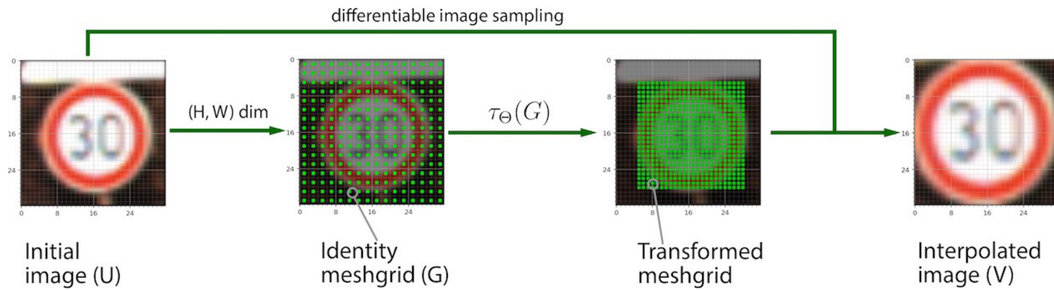
8.5. Spatial Transformers

A konvolúciós hálózatok egyik legnagyobb hiányossága, hogy a geometriai transzformációk nagy részére nem invariánsak. Ennek egyik lehetséges megoldása, hogy a transzformációkat tanulható módon beleépítjük a hálóba. Erre az alapötletre épülnek az ún. Spatial Transformer architektúrák. Ennek az architektúrának a lényege, hogy a réteg bemenetére kapott 2/3D aktivációs tömböt valamilyen konzisztens formába transzformálja. Ehhez először a rétegnek meg kell határoznia azt a geometriai transzformációt, ami ezt a konzisztens formába történő átalakítást az adott bemenetre elvégzi. Ezt egy lokalizációs háló végzi, amely kimenetként előállítja a transzformáció paramétereit.



8.14. ábra. *A Spatial Transformer felépítése.*

A réteg működéséhez fontos, hogy a transzformáció maga deriválható legyen a paraméterei szerint, hogy a lokalizációs háló súlyai a backpropagation segítségével tanulhatók legyenek. Ehhez a Spatial Transformer réteg egy mintevételezésre alapuló technikát alkalmaz: A transzformáció paramétereinek segítségével először egy egyenletes rácsháló koordinátáit transzformálja (ez deriválható). Ezt követően a bemeneti tömb elemeit az új rácsháló koordinátáinak megfelelő helyre mozgatja bilineáris interpoláció segítségével (ez szintén deriválható). Így a transzformáció csupa deriválható művelet kompozíciójaként áll elő, következésképp ő maga is deriválható lesz.



8.15. ábra. *A deriválható mintavételezés.*

További Olvasnivaló

[1] Jeff Heaton. „Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning". *Genetic Programming and Evolvable Machines* 19.1-2 (2017. okt.), 305–307. old. DOI: 10.1007/s10710-017-9314-z. URL: https://doi.org/10.1007%2Fs10710-017-9314-z.

[28] Christopher B. Choy és tsai. „3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction". *Computer Vision – ECCV 2016*. Springer International Publishing, 2016, 628–644. old. DOI: 10.1007/978-3-319-46484-8_38. URL: https://doi.org/10.1007%2F978-3-319-46484-8_38.

[29] David Eigen és Rob Fergus. „Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-scale Convolutional Architecture". *2015 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2015. dec. DOI: 10.1109/iccv.2015.304. URL: https://doi.org/10.1109%2Ficcv.2015.304.

[30] Daniel Maturana és Sebastian Scherer. „VoxNet: A 3D Convolutional Neural Network for real-time object recognition". *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015. szept. DOI: 10.1109/iros.2015.7353481. URL: https://doi.org/10.1109%2Firos.2015.7353481.

[31] Yin Zhou és Oncel Tuzel. „VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection". *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, 2018. jún. DOI: 10.1109/cvpr.2018.00472. URL: https://doi.org/10.1109%2Fcvpr.2018.00472.

[32] Hang Su és tsai. „Multi-view Convolutional Neural Networks for 3D Shape Recognition". *2015 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2015. dec. DOI: 10.1109/iccv.2015.114. URL: https://doi.org/10.1109%2Ficcv.2015.114.

[33] Charles R. Qi és tsai. *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*. 2017. eprint: 1612.00593. URL: http://www.arxiv.org/abs/1612.00593.

[34] Yangyan Li és tsai. *PointCNN: Convolution On $\mathcal{X}$-Transformed Points*. 2018. eprint: 1801.07791. URL: http://www.arxiv.org/abs/1801.07791.

[35] Roman Klokov és Victor Lempitsky. *Escape from Cells: Deep Kd-Networks for the Recognition of 3D Point Cloud Models*. 2017. eprint: 1704.01222. URL: http://www.arxiv.org/abs/1704.01222.

[36] Yutong Feng és tsai. *MeshNet: Mesh Neural Network for 3D Shape Representation*. 2018. eprint: 1811.11424. URL: http://www.arxiv.org/abs/1811.11424.

[37] Max Jaderberg és tsai. *Spatial Transformer Networks*. 2016. eprint: 1506.02025. URL: http://www.arxiv.org/abs/1506.02025.

III. rész

Felügyelet Nélküli Tanulás

9. fejezet

Generatív Hálók

9.1. Bevezetés

A neurális hálók egyik leggyakoribb felhasználása a bemenetként kapott kép osztályozása, szegmentálása, illeve a képminőség javítása. Ezekben a közös vonás az, hogy a bemenet már egy kész kép, amiből csak információkat kell kinyerni. Azonban léteznek olyan neurális hálózatok, amelyek ennél többre is képesek: létre tudnak hozni (*generálni tudnak*) egy teljesen új képet akár értelmetlen, véletlen zaj bemenetből is, vagy újra tudnak alkotni egy meglévő képet egy másik stílusban. A most következő előadásban tehát a neurális hálók segítségével történő képgenerálásról lesz szó.

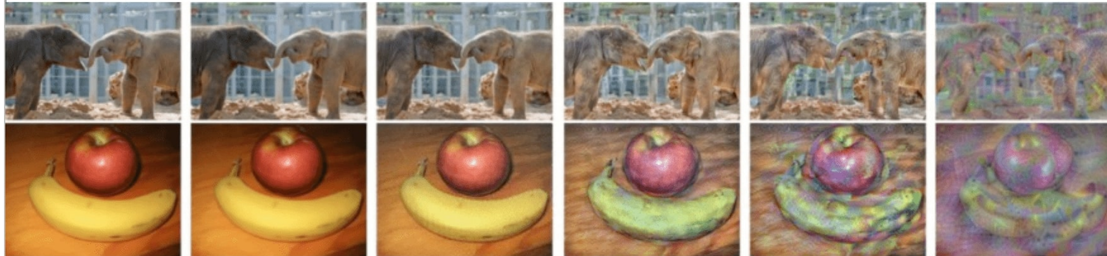
9.2. Stílus átültetés

A művészet és szórakoztatóipar terén hozott érdekes áttörést az úgynevezett neurális stílus átültetés, melynek során a neurális háló egy tetszőleges bemeneti képet képes volt újragenerálni egy adott festmény/műalkotás stílusában. Ennek az eljárásnak a megértéséhez azonban röviden meg kell tárgyalnunk két másik vizsgálati módot: a jellemző rekonstrukciót és a textúra szintézist.

9.1. ábra. *A neurális stílusátültetés feladata.*

9.2.1. Jellemző rekonstrukció

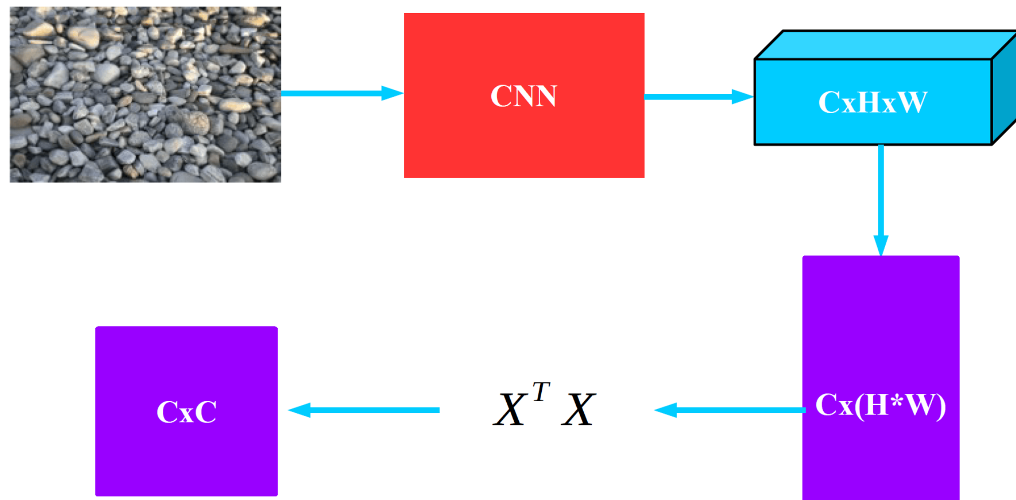
A jellemző rekonstrukció célja a neurális háló egyes rétegeinek absztrakciós képességének vizsgálata volt. A rekonstrukció megfogalmazott célja, hogy ha adott egy kép, amit előreküldtünk a hálón egy adott rétegig, és rendelkezésre állnak ezek aktivációi, akkor állítsuk elő azt a képet, amely pont ugyanazokat az aktivációkat okozza a neurális hálóban. Ehhez természetesen ugyanúgy a guided backpropagation műveletét használjuk, csak a kimeneti gradiensek előírása helyett hibafüggvényt használunk, ami az aktivációk egyezőségét biztosítja (pl. négyzetes hiba). Fontos még megjegyezni, hogy itt ugyanúgy szükséges a kép valószerűségének biztosítása szűrések és regularizáció segítségével.



9.2. ábra. *A jellemző rekonstrukció eredménye. Látható, hogy az egyre mélyebben lévő rétegek esetén egyre absztraktabb rekonstrukciót kapunk.*

9.2.2. Textúra szintézis

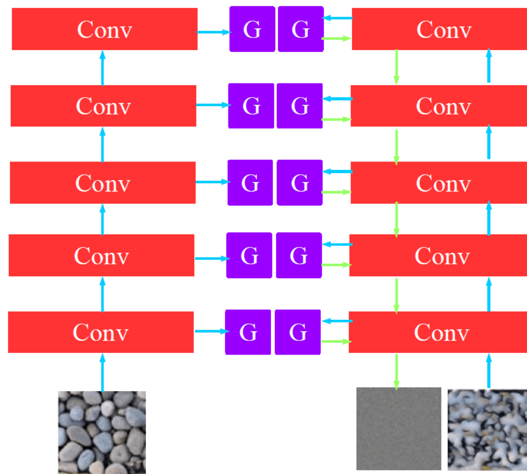
A textúra szintézis lényege, hogy egy valós kép segítségével készíteni tudjuk egy szintetikus képet, amely az eredeti képhez hasonló textúrával rendelkezik. Ehhez azonban először szükségünk van arra, hogy a textúrát leíró matematikai mennyiséggel megismerkedjünk. A textúrát első sorban az egyes pixelek/képrészletek statisztikai kapcsolatai határozzák meg. Valószínűségi változók közti kapcsolatokat pedig elsősorban a kovariancia mátrix segítségével szoktunk jellemezni. Éppen ezért a textúra szintézis során előállítunk egy ehhez hasonló mennyiséget, amelyet a Gram-mátrixnak nevezünk. A Gram-mátrixot egy adott réteghez tudjuk előállítani, és a rétegben lévő csatornák kovarianciamátrixával arányos.



9.3. ábra. *A Gram-mátrix számítása egy adott rétegre.*

A textúra szintézis után a kiválasztott képet egy előre betanított hálón előreküldjük, majd kiszámoljuk a Gram-mátrixok értékeit az egyes rétegekre. Ezt követően a hálón a véletlen zajjal inicializált képet küldünk előre. A backpropagation során ezúttal azt írjuk elő, hogy az egyes rétegeknél keletkező Gram-mátrixok legyenek egymáshoz közel négyzetes értelemben. Így a szintetikus képet a konvergenciáig iteratívan módosítva állítjuk elő a kívánt textúrát.

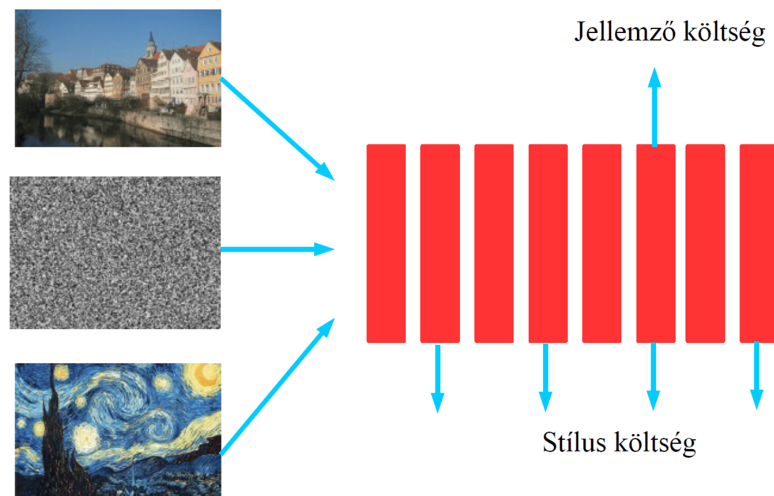
$$E_l = \|G_l^1 - G_l^2\|^2$$



9.4. ábra. *A textúraszintézis algoritmusa.*

9.2.3. Neural Style Transfer

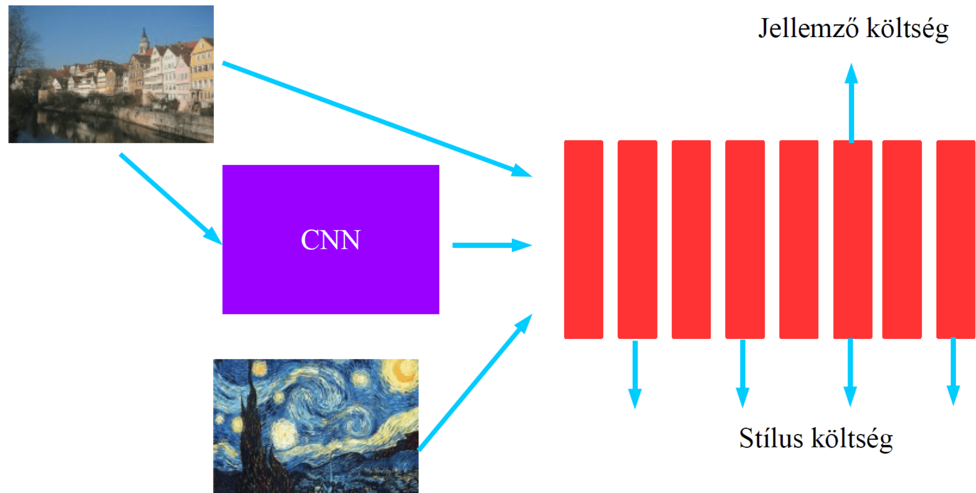
Ezek után könnyen látható, hogy a neurális stílus átültetés nem más, mint az előző két módszer kombinációja. Kiválasztunk a betanított hálón egy réteget, amire előírjuk a jellemző rekonstrukció költségét, a textúra rekonstrukció költségét pedig az összes rétegre előírva megkaphatjuk a kívánt képet. A módszer egyik nagy hátránya, hogy az átültetett kép generálásához számos forward és backward lépést kell végrehajtani, ami meglehetősen lassú működést eredményez.



9.5. ábra. *A neurális stílusátültetés alap algoritmusa.*

Érdemes azonban észrevenni, hogy amennyiben az előző módszerrel előállt egy végeredmény, akkor ez felhasználható egy tanító adatbázis készítésére, aminek segítségével egy konvolúciós háló betanítható egy adott stílus tetszőleges képre történő átültetésére. Ennek a módszernek azonban még egy megmaradó problémája, hogy különböző stílusokhoz külön hálókat kell betanítani.

Érdekes felfedezés volt azonban, hogy be lehet tanítani egyetlen hálót több stílus átültetésére is a paraméterszám megtöbbszörözése nélkül. Ehhez csupán annyit kellett tenni, hogy a hálókban alkalmazott instance normalizációs rétegeknek az összes alkalmazni kívánt stílushoz külön-külön paramétereket tanítottak, míg a konvolúciós rétegek az összes stílus esetében azonos paraméterekkel rendelkeztek. Ennek nagy előnye, hogy egy konvolúciós rétegnek több száz és több millió közt



9.6. ábra. *A neurális stílusátültetés megtanítása egy neurális hálónak.*

mozog a paraméterszáma, míg egy normalizációs rétegnek általában 2, vagy 4 paramétere van. Éppen ezért nem gond, hogy ezekből minden stílushoz külön értékeket kellett tanulni.

A több stílust átültetni képes hálók képesek arra is, hogy átmenetet alkossanak két, vagy több stílus között, ehhez nem kell mást tenni, csak a külön paraméterek között interpolációt végezni.

9.7. ábra. *A stílusok közti átmenet.*

9.3. Generatív modellek

Amint azt láttuk, az eddig tárgyalt hálóarchitektúrák vizualizációja megoldható, valamint ezen hálók limitált mértékben alkalmasak lehetnek képszintézisre is. Ez a képességük azonban korlátos, ugyanis az eddig tárgyalt tanítási módszerek úgynevezett diszkriminatív neurális hálókat eredményeznek. Ez azt jelenti, hogy a neurális hálók csak azt tanulják meg, hogy az elvárt kimenet

hogyan függ a bemenettől, például hogy miben különbözik két osztály egymástól vizuálisan. A valószínűségszámítás nyelvén megfogalmazva ez azt jelenti, hogy a diszkriminatív algoritmusok a be- és kimenet közti $P(y|x)$ feltételes valószínűségi függvényt tanulják meg.

Ez azonban azt jelenti, hogy a bemenetek eloszlásáról (vagyis arról, hogy konkrétan hogyan néznek ki az ismert osztályok) semmit nem tudnak. Ahhoz, hogy a neurális háló a képek kinézetét is megtanulja, a $P(y, x)$ együttes sűrűségfüggvényt kellen megtanulni. Ennek a következménye, hogy a képek eloszlása is ismert, így pedig lehetőségünk nyílik arra, hogy ebből az eloszlásból véletlenszerűen mintavételezve valószerű képeket generáljunk. Az ilyen módon tanított hálókat ezért generatív modelleknek nevezzük. Érdemes megjegyezni, hogy az együttes sűrűségfüggvény bonyolultabb, így bár egy generatív modell is használható osztályzásra, a diszkriminatív modellek általában jobban teljesítenek.

Generatív modelleket akkor is tudunk tanítani, ha nem állnak rendelkezésre címkék, ilyenkor egyszerűen csak a bemenetek eloszlását $P(x)$ tanítjuk meg a hálónak. Ebben az esetben felügyelet nélküli generatív tanulásról beszélhetünk.

9.3.1. PixelRNN

A generatív modellek közül a legegyszerűbb a PixelRNN, amely egy egyszerű visszacsatolt neurális háló, amely a kép pixeleit sorrendben képes generálni. A PixelRNN a kép valószínűségét közvetlenül modellezi az alábbi módon:

$$P(Image) = \prod_{i=1}^{n} p(x_i|x_1, \ldots, x_{i-1}) \tag{9.1}$$

vagyis a következő pixel valószínűsége az előző körökben generáltaktól függ, a háló feladata pedig a teljes tanító adatbázis valószínűségének maximalizálása. Ez a gyakorlatban úgy valósul meg, hogy a háló megkapja a kép első pixelét, amelynek alapján elvárjuk, hogy pontosan becsülje meg a másodikat. Ezt követően a háló megkapja a bemenetére a második pixelt is, és ez alapján megbecsli a harmadikat, és így tovább. A generálás során a háló kap egy kiinduló pixel értéket, majd ez alapján generálja a következőt. Ezt követően minden lépésben az előző generált pixel kerül a bemenetére, és így készül el a teljes kép.

Érdemes megjegyezni, hogy a PixelRNN működéséhez szükség van valamilyen előre meghatározott pixelsorrendre, aminek alapján a képen végigmegyünk. A PixelRNN egyik nagy hátránya, hogy a pixelenként történő képgenerálás rendkívül lassú, ami a tanítást és a futtatást is költségessé teszi. A PixelRNN-nek létezik konvolúciós változata is, amely valamelyest javít a generálás sebességén.

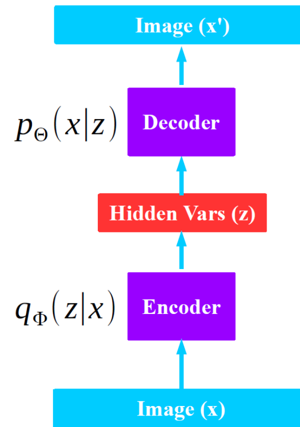
9.3.2. Variációs Autoencoder

A Variációs Autoencoder (VAE) névre hallgató algoritmusok egy eltérő filozófia mentén képesek képeket generálni. Az alapöltetük az, hogy léteznek valamilyen z ismeretlen, rejtett (látens) változók, amik kompakt módon képesek leírni a kép tartalmát. A variációs autoencoder célja, hogy ezekből a látens változókból tanulja meg legenerálni a hozzájuk tartozó képet. Fontos tudni, hogy a VAE betanítható anélkül, hogy a tervezők tudnák, hogy pontosan mik ezek a látens változók.

$$P_\Theta(Image) = \int p_\Theta(z) p_\Theta(x|z) dz \tag{9.2}$$

A Variációs Autoencoder tárgyalása előtt azonban célszerű röviden bemutatni az elődjét, a hagyományos Autoencodert. Ezek a neurális háló modellek a 2000-es évek végén voltak népszerűek, amikor a 9. előadásban ismertetett eljárások ismerete nélkül nem voltunk képesek mély neurális hálókat megbízható módon betanítani. Az autoencoder lényege, hogy a bemenetére kapott vektort

egy belső rejtett rétegre képzi le. Fontos azonban, hogy ez a rejtett réteg kevesebb változót tartalmaz, mint a bemenet. Ezt követően az Autoencoder feladata, hogy a kimenetén ebből a rejtett rétegből minél kisebb négyzetes hibával állítsa elő az eredeti bemenetet (nagyobb méretű rejtett réteg esetén ez ugye triviálisan egyszerű lenne).



9.8. ábra. *Az Autoencoder felépítése.*

Az Autoencoder állhat természetesen több rejtett rétegből is, ekkor a rétegeket külön-külön adjuk hozzá az architektúrához, mindegyiket arra tanítva, hogy az előző réteg aktivációit legyen képes visszaállítani. Egy ilyen Stacked Autoencoder-nek is nevezett architektúrát két részre oszthatunk: a leskálázó encoder részre, amely a bemenetet leképezi a rejtett változók terébe, valamint a felskálázó dekóder részre, amely a rejtett változókból állít elő bemenetet. A 2000-es évek elején az encoder részt gyakorta használták osztályozási feladatok megoldására.

Felmerülhet bennünk, hogy használjuk a dekóder hálót arra, hogy valamilyen véletlen rejtett változó vektorból képet szintetizáljunk. Ezzel azonban van egy nagy probléma: ahhoz, hogy valószerű képhez tartozó rejtett változókat generáljuk, ismerni kéne azoknak az eloszlását. Sajnos az Autoencoder esetében nem feltételezhetjük, hogy a látens változók normális eloszlásúak. Ezért felmerülhet bennünk az öltet, hogy valamilyen módon próbáljuk a tanítás során rákényszeríteni a sztenderd normális eloszlást a látens változókra. Ehhez felhasználhatjuk a korábban már megismert KL-divergencia fogalmát, amely egy p_1 és p_2 eloszlás esetében az entrópia és a keresztentrópia különbsége, vagyis a két eloszlás hasonlósági mércéje:

$$D_{KL}(p_1||p_2) = E_x[log\frac{p_1}{p_2}] \quad (9.3)$$

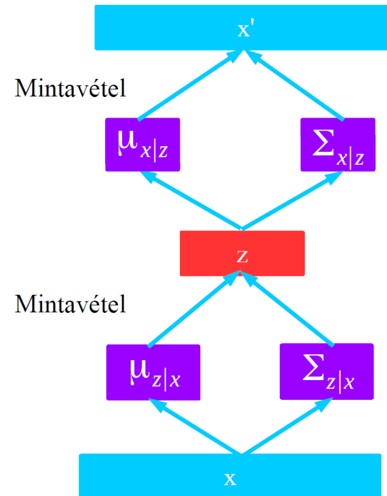
A Variációs Autencoderek tanítása során a költségfüggvény két részből áll: az első az Autoencoderekből ismert négyzetes rekonstrukciós hiba, míg a másik a KL-divergencia az előírt és a háló által generált eloszlások között.

$$L_i(\Theta, \Phi) = E_x[logp_\Theta(x_i|z)] - D_{KL}(q_\Phi||p(z)) \quad (9.4)$$

ahol p_Θ a dekóder által reprezentált eloszlás (rekonstrukciós hiba), q_Φ az encoder eloszlása, $p(z)$ pedig a látens változók előírt eloszlása.

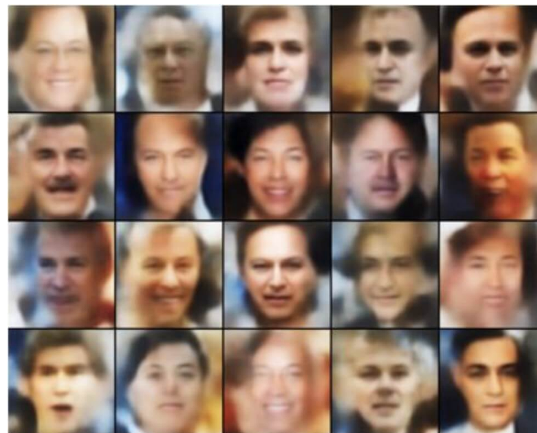
Érdemes megjegyezni, hogy a VAE hálók nem közvetlenül a látens változó értékeket és a rekonstruált kimenetet állítják elő, hanem ezeknek a várható értékét és kovarianciamátrixát. A tanítás és a tesztelés során is ezekből mintavételezünk, és a hálót az így generált értékekkel tanítjuk.

Azt is fontos tudni, hogy a rejtett változóra alkalmazott KL-divergencia kritériumát egy minibatchen belül az összes képhez generált látens változók együttes eloszlására alkalmazzuk, nem pedig képenként. Ez azért fontos, mert azt szeretnénk, ha a látens változók eloszlása az egész adatbázisra legyen normális eloszlású, de ne legyen minden külön képre ugyanaz (különben nem is tudnánk a rekonstrukciót elvégezni).



9.9. ábra. *A variációs autoencoder felépítése.*

Az így kapott Variációs Autoencoder egy jól használható eljárás, ami a PixelRNN-es megoldásnál lényegesen gyorsabb, azonban a generált képek meglehetősen homályosak, még a konvolúciós VAE megoldások esetén is.



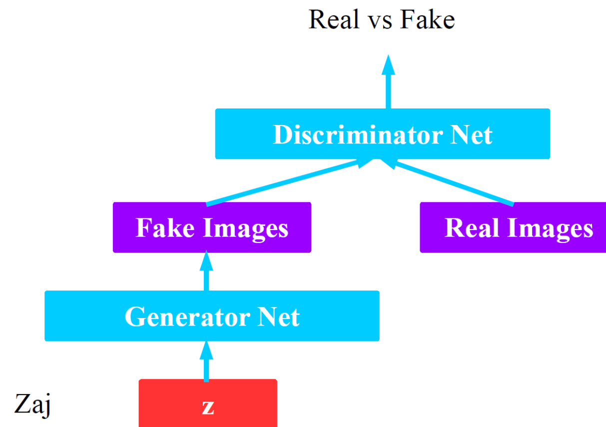
9.10. ábra. *Egy Variációs Autoencoder által generált arcképek.*

9.3.3. GAN

A generatív modellek koronázatlan királya a Generative Adversarial Networks (GAN) névre hallgató megoldás. Ez a architektúracsalád a valószínűségszámítás helyett a játékelmélet módszereiből nyerte az alapötletét: két egymás ellen versengő neurális háló az alapja. Az egyik a diszkriminátor háló, amely alapvetően egy leskálázó jellegű bináris osztályozó háló. A diszkriminátor feladata, hogy egy, a bemenetére kapott képről eldöntse, hogy az valódi-e vagy generált hamisítvány. Az ellenfele a generátor háló, ami egy felskálázó jellegű háló, ami a bemenetére adott véletlen zajszerű látens változó vektorból igyekszik olyan képet generálni, amivel képes a diszkriminátor hálót megtéveszteni.

A GAN-ok tanítása során a két háló költségét egyszerre optimalizáljuk az alábbi módon:

$$\begin{gathered} max_{\Theta_d}[E_x log D_{\Theta_d}(x) + E_z log(1 - D_{\Theta_d}(G_{\Theta_g}(z)))] \\ min_{\Theta_g}[E_z log(1 - D_{\Theta_d}(G_{\Theta_g}(z)))] \end{gathered} \tag{9.5}$$



9.11. ábra. *A GAN hálók architektúrája.*

A diszkriminátor háló tehát egyszerre próbálja maximalizálni a saját kimenetét a valós képeken és a kimenetének inverzét a generált képeken. Közben a generátor háló pedig ezt a második tagot igyekszik minimalizálni. Ehhez az SGA (Stochastic Gradiens Ascent), vagyis a gradiens maximalizálás módszerét alkalmazzuk a diszkriminátoron, és a szokásos SGD módszert a generátoron. Ezzel viszont adódik egy jelentős probléma: a logaritmus alakja miatt a költségfüggvény gradiense kicsi akkor, ha a generált képek nagyon rosszak: márpedig a tanítás elején általában ez a helyzet. Ezért a költségfüggvényt úgy módosítjuk, hogy a generátor ne a lebukás esélyét próbálja minimalizálni, hanem a sikeres átverés esélyét maximalizálja. Így a generátoron is SGA módszert alkalmazunk. Ennek hatására a költségfüggvények az alábbi módon változnak:

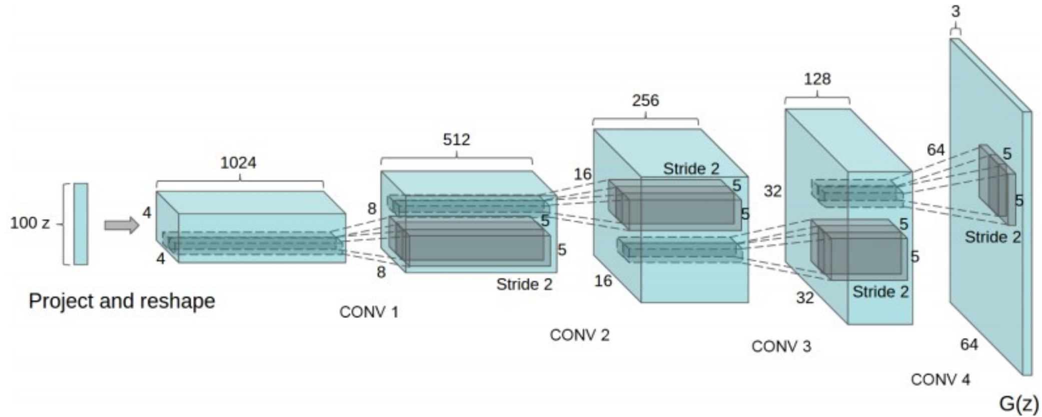
$$\begin{gathered} max_{\Theta_d}[E_x log D_{\Theta_d}(x) + E_z log(1 - D_{\Theta_d}(G_{\Theta_g}(z)))] \\ max_{\Theta_g}[E_z log(D_{\Theta_d}(G_{\Theta_g}(z)))] \end{gathered} \tag{9.6}$$

A GAN-ok tipikus tanítási ciklusa az alábbi módon alakul:

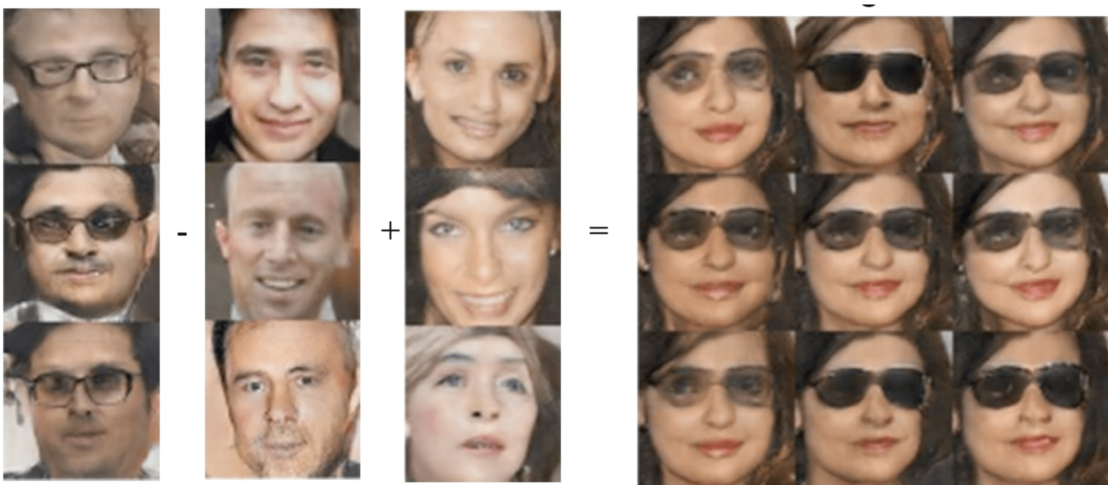
1. k lépésig:
   (a) Egy minibatch generált kép előállítása
   (b) Egy minibatch valódi kép elkérése
   (c) Diszkriminátor tanítása
2. Egy minibatch generált kép előállítása
3. Generátor tanítása

A GAN hálózatok nagy sikernek örvendenek a mai napig képek generálásánál. Ennek az egyik oka, hogy könnyedén lehet belőlük konvolúciós verziót készíteni, ezeket általában DCGAN-nak (Deep Convolutional GAN) nevezzük. Ezek a hálózatok általában strided konvolúciós réteg segítségével oldják meg a le- és felskálázást. Felskálázás esetén természetesen a konvolúció transzponált. Fontos még a batch normalization használata és az aktivációs függvény megválasztása, a GAN hálók ugyanis gyakorta küszködnek konvergenciaproblémával. A diszkriminátor általában Leaky ReLU-t használ, míg a generátor közönséges ReLU-t.

Fontos megemlíteni, hogy a GAN hálózatok által felhasznált látens változók valóban képesek a generált képek bizonyos tulajdonságait az ember számára logikus módon leírni. Erre jó példa az, hogy a látens változó vektorokon alkalmazott matematikai műveletek az emberi logikának megfelelően működnek, ami a nyers képekre egyáltalán nem igaz. Ezen felül a látens változók terében két tetszőleges pont között interpolálva érdekes módon az egyes képek tartalma közötti szemantikai átmeneteket kapjuk.



9.12. ábra. *A DCGAN generátor felépítése.*



9.13. ábra. *A GAN által generált képek esetén a látens változók terében végzett matematikai műveletek szemantikailag értelmes eredményt adnak.*

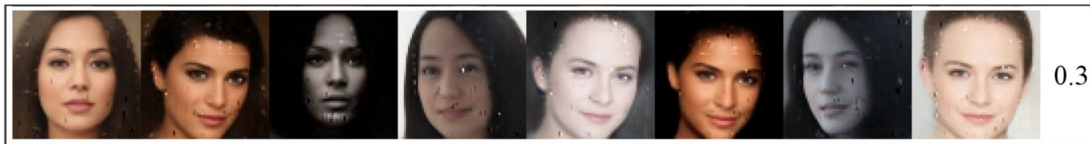
A GAN hálók habár rendkívüli minőségű képeket képesek generálni, sok gyakorlati nehézségtől szenvednek. Nehéz például a fent említett rejtett változók jelentését kitalálni, mivel az emberek számára értelmezhető szemantikus változók a GAN látens terében általában más változókkal "összegabalyodva" jelennek meg. Ez alatt azt értjük, hogy egy változó nem csupán a szemüveg jelenlétét, a mosolygást, vagy a hajszínt tartalmazza, hanem ezek valamilyen kombinációját.

A GAN-ok tanítása ráadásul rendkívül nehéz, ugyanis a hatékony tanuláshoz az szükséges, hogy a két hálórész közti versengést egyik fél se nyerje meg, ekkor ugyanis leáll a tanulás. Ennek a külön esete az ún. Mode Collapse, amely azt a jelenséget takarja, amikor a diszkriminátor megnyeri a játékot, és képes az adatbázisban lévő összes képet felismerni. Ekkor a generátor igyekszik rátanulni arra a néhány példára, amit a diszkriminátor éppen valószerűnek tart. Ennélfogva a generátor minden bemenethez ugyanazt a képet (vagy legalábbis nagyon hasonló képeket) fog készíteni.

Ennek a jelenségnek az elkerülésére alkották meg a feltételes (Conditional) GAN, vagyis a CGAN hálózatokat, amelyek a címkék használatát vezetik vissza. A CGAN esetében mindkét háló a bemenetére a kép/zaj mellé a címkét is megkapja, így a generátornak bizonyos osztályba tartozó képeket kell tudni készíteni. Ez továbbá könnyedén lehetővé teszi a kívánt osztályba tartozó képek generálását. Ebben az esetben viszont címkézett adatbázis szükséges a háló tanításához.

Egy valamivel fejlettebb változat az InfoGAN, melynek esetében a generátor egy véletlenszerűen generált kategorikus címkét is kap a bemeneti véletlen zaj (látens változó) mellé. Az így generált képből a diszkriminátornak nemcsak az a feladata, hogy eldöntse, hogy a kép valódi-e, hanem az is, hogy visszafejtse ezeket a kategorikus címke változókat, vagyis hogy megbecsülje az egyes

9.14. ábra. *Egy DCGAN háló által generált beltéri képek (látens vektortérben véletlen pontok közt interpolálva).*



9.15. ábra. *A mode collapse jelensége. Megfigyelhető a hasonlóság az azonos színnel aláhúzott képek közt.*

címke kategóriák valószínűségét a generált képből. Fontos, hogy a tanítás során a generátort is arra tanítjuk, hogy a címke változók eltalálhatók legyenek, így garantálni tudjuk, hogy a generált kép ezektől függően változzon, így elkerülve a mode collapse jelenségét. Ez a gyakorlatban úgy történik, hogy a költségfüggvénybe bekerül a címke és a generált kép közötti kölcsönös információ negatív előjellel:

$$\mathcal{L}_{\text{infoGAN}}(G, D) = \mathcal{L}_{\text{GAN}}(G, D) - \lambda\, I(c, G(z, c)) \tag{9.7}$$

ahol az I kölcsönös információ értéke nulla, ha a címke c és a generált kép $G(z, c)$ függetlenek/irrelevánsak egymásnak, és egy magas érték, ha összefüggenek.

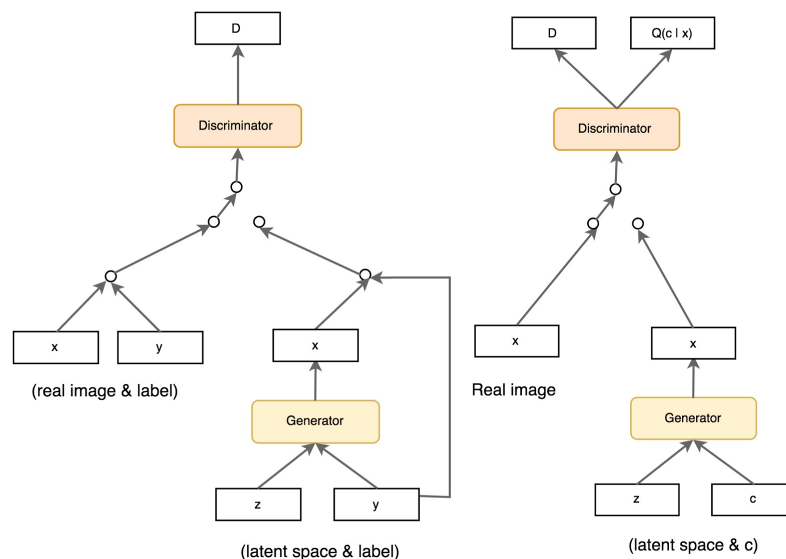
A következőekben néhány további érdekes GAN variáns kerül bemutatásra.

BEGAN

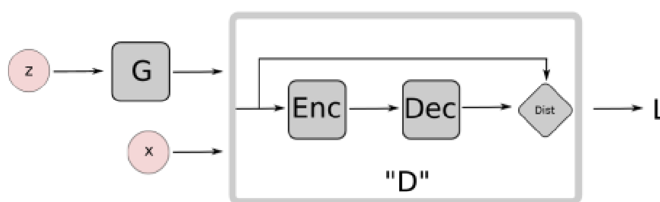
A Boundary Equilibrium GAN, vagy BEGAN, voltaképp az Autoencoder és a GAN ötletét vegyíti egybe. A BEGAN architektúra esetében a diszkriminátor egy Autoencoder, amelytől azt várjuk el, hogy a valódi képekre pontosan visszaállítsa az eredeti képet, míg a generált képek esetében rossz rekonstrukciót adjon. Ez a megoldás a kutatások szerint lényegesen szebb képeket képes alkotni.

ProGAN

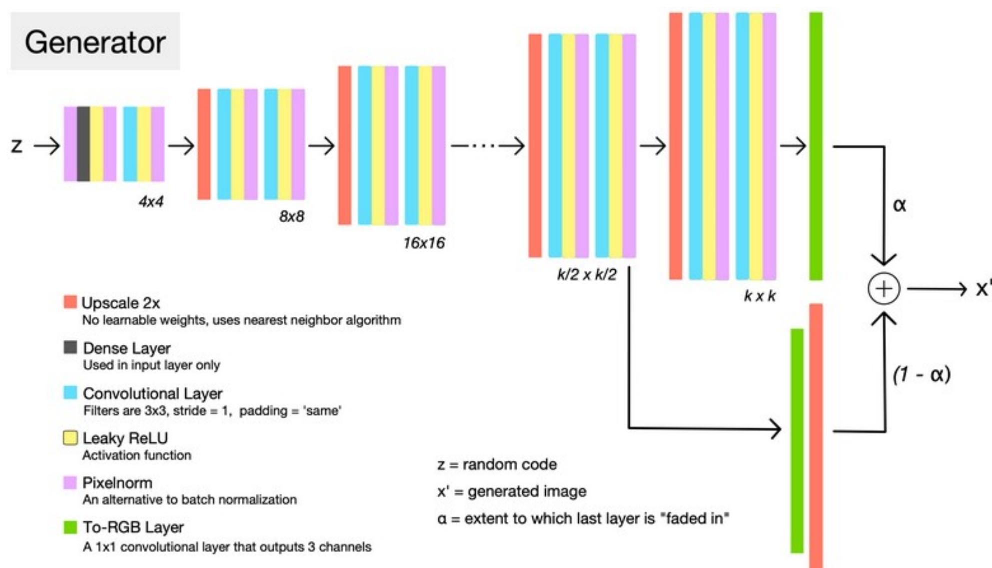
A ProGAN alapelve, hogy a valószerű képek generálását fokozatosan, kis felbontásból kiindulva egyre finomítva tanulja meg. A háló először 4×4 képeket tanul meg generálni, majd a meglévő generátorhoz és diszkriminátorhoz újabb szinteket hozzáadva egyre nagyobb felbontásban tanul meg képet generálni. Ennek előnye a hierarchikus, egyre részletesebb tudás felépítése. A tanulást nagyban segíti, hogy a generátor aktuális kimenete úgy áll elő, hogy az előző már betanult réteg kimenetét fix felskálázás után hozzáadjuk a háló utolsó rétegének kimenetéhez.



9.16. ábra. *A CGAN (bal) és az InfoGAN (jobb) hálózatok architektúrája.*



9.17. ábra. *A BEGAN modell.*

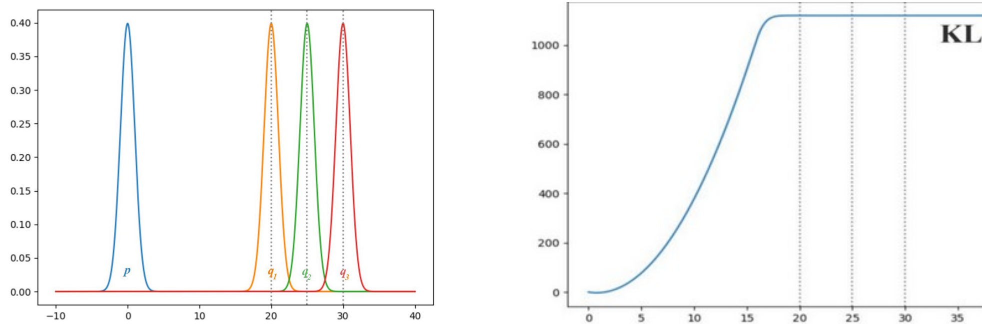


9.18. ábra. *A ProGAN modell.*

WGAN

A Wasserstein GAN, vagy WGAN modell abban különbözik a hagyományos GAN-tól, hogy az osztályzáshoz a hagyományos keresztentrópia költségfüggvényt egy másikra cseréli. Emlékezzünk, hogy a kereszt-entrópia költség közeli rokonságban áll az ún. KL-divergenciával, ami valószínűségi eloszlások hasonlósági mércéjeként értelmezhető. Ennek a költségfüggvénynek jelentős hátránya azonban a már korábban is felmerülő eltűnő gradiensek problémája: a KL-divergencia ugyanis túl

nagy eltérések esetén szaturálódik (vagyis a függvény alakja lapos lesz), így a deriváltja nulla lesz.



9.19. ábra. *A KL-divergencia alakulása. A bal oldalon 4 eloszlás található, a jobb oldalon pedig a p-vel és a q-val jelölt eloszlások közti KL divergencia alakulása a várható értékek különbségének függvényében. Látható, hogy 15 felett a KL-divergencia kilaposodik.*

Célszerű lehet tehát a KL-divergencia helyett egy másik hasonlósági mérce bevezetése. Ez az alternatív függvény a Wasserstein, vagy más néven a földmozgató (Earth movers) távolság. Ez a távolság mérce rendkívül szemléletes: képzeljünk el dobozokat, melyek különböző pozícióban helyezkednek el. Feladatunk, hogy ezeket a dobozokat előre meghatározott célpozíciókba mozgassuk. Az azonban mindegy, hogy melyik dobozt hova mozgatjuk, egészen addig, amíg minden célpozícióban a megfelelő mennyiségű doboz van. Egy doboz egy egységgel történő mozgatásának fix költsége van, így a teljes átrendezést különböző összköltségű stratégiákkal tudjuk megoldani. Nyilvánvalóan létezik (legalább) egy stratégia, amelynek a teljes költsége minimális. Ezt a költséget nevezzük Wasserstein-távolságnak.

CycleGAN

A CycleGAN egy nagyon népszerű GAN architektúra, amelyet különböző képstílusok közötti átalakításokra használnak. Például a CycleGAN képes megtanulni

- művészi képek valósághű képekké való átalakítását és fordítva,
- ló képek zebra képekké való átalakítását és fordítva,
- téli képek nyári képekké való átalakítását és fordítva,
- emberi arcok különböző korcsoportúvá alakítását (lást FaceApp mobil alkalmazás).

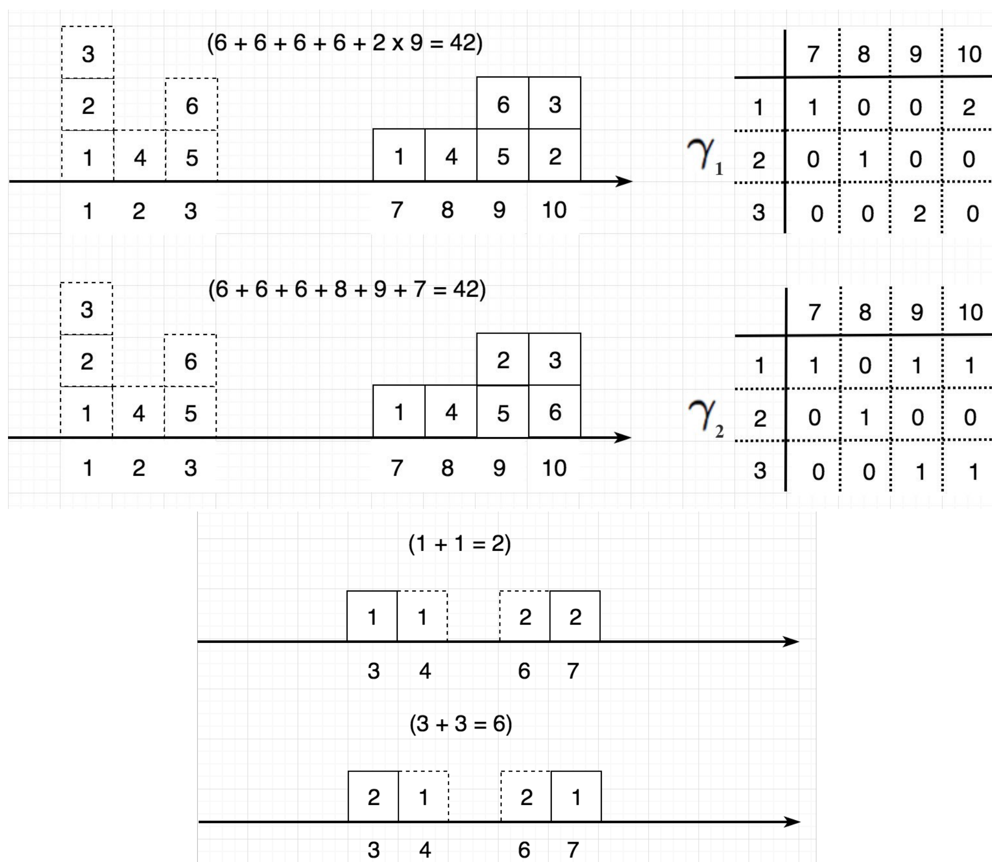
Tegyük fel például, hogy X a ló képek halmaza, Y pedig a zebra képek halmaza. A ló képek zebra képekké átalakításához a cél egy olyan $G : X \to Y$ leképezési függvény megtanulása, amely esetén a $G(X)$ által generált képek megkülönböztethetetlenek az Y képektől. Ennek megvalósulásáról az úgynevezett ellenséges (adversarial) költségfüggvény gondoskodik. Ezzel a költségfüggvénnyel nemcsak a G generátor leképezés kerül tanításra, hanem egy $F : Y \to X$ inverz leképezés is. Ez két külön GAN modellt jelent: az egyik a G generátorból és egy D_Y diszkriminátorból áll, a másik pedig az F generátorból és egy D_X diszkriminátorból. A CycleGAN emellett cikluskonzisztencia (cycle consistency) költséget is használ az $F(G(X)) = X$, illetve $G(F(Y)) = Y$ szabályok érvényesítésére, vagyis a rekonstrukciós hiba minimalizálására. Ennek hátterében az áll, hogy a rekonstrukciós hiba minimalizálása a két adathalmaz eloszlása közti különbség minimalizálását is jelenti.

Tehát összességében a CycleGAN egy háromelemű költségfüggvényt használ:

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\mathrm{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\mathrm{GAN}}(F, D_X, Y, X) + \lambda \mathcal{L}_{\mathrm{eye}}(G, F) \tag{9.8}$$

ahol az első tag a G generátorral és D_Y diszkriminátorral rendelkező GAN költségfüggvénye, a második tag a D_X diszkriminátorral rendelkező GAN költségfüggvénye, a harmatik tag pedig a cikluskonzisztencia költségvüggvénye. Az optimalizáció pedig a következőképpen írható fel:

$$G^*, F^* = \arg\min_{G,F} \max_{D_X, D_Y} \mathcal{L}(G, F, D_X, D_Y) \tag{9.9}$$



9.20. ábra. *Az átrendezési stratégia (felül). A dobozok jelenlegi helyzetét a szaggatott vonallal jelölt keretek adják, míg a kívánt helyüket a teljes vonalas részek jelölik. Látható, hogy az ábrán megadott mozgatási stratégia költsége pont 42. Az alsó ábra illusztrálja, hogy nem minden mozgatási stratégia ekvivalens.*

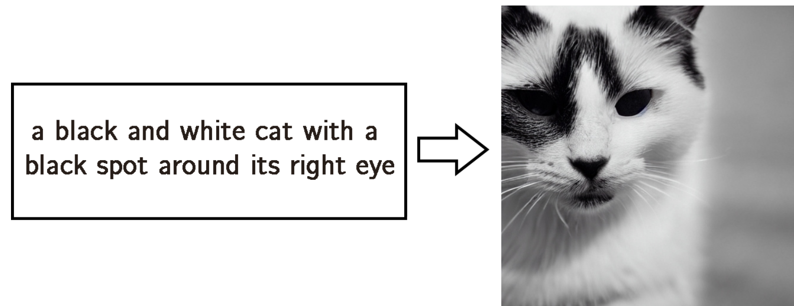
Érdemes megemlíteni a közelmúltban történő fiaskót, ami rendkívül jól illusztrálja a deep learning algoritmusok tanítása közben elkövethető hibákat. A CycleGAN alkalmazása az volt, hogy légi felvételekből képes legyen térképet készíteni, majd az elkészített térképekből képes legyen négyzetes értelemben minél pontosabban visszaállítani az eredeti felvételt. A kutatóknak feltűnt, hogy az algoritmus túlságosan jó volt: olyan részleteket is pontosan vissza tudott állítani az eredeti képről, ami a generált térképen nem szerepelt (fák, évszak, parkoló autó). A CycleGAN ugyanis csalásra vetemedett: az emberi szem számára láthatatlan (a négyzetes hiba szempontjából pedig elhanyagolható) módon a generált térképben elrejtette az eredeti műholdfelvétel információit, így pedig ezeket a részleteket is könnyedén vissza tudja állítani. A neurális háló tehát voltaképpen feltalálta a szteganográfia módszerét.

T2I (text-to-image)

Ha egy GAN-t macskaképek generálására képeznek ki, akkor a modell nagyon jól fog tudni valósághű (két szemű, két fülű, bajszos) macskákat létrehozni, a macska színe és mintája azonban teljesen véletlenszerű lesz, Ha egy üzleti felhasználási esetben például kifejezetten olyan macskák képére lenne szükségünk, amelyek fekete-fehérek és van egy fekete foltjuk a jobb szemük körül, akkor nagyon nehéz elérni, hogy a GAN ezt a kérésünket teljesítse.

A T2I (text-to-image) egy olyan GAN architektúra, amely jelentős előrehaladást ért el az explicit szöveges leírás alapján történő (értelmes) képek generálásában. Ez a GAN modell a bemeneteként egy szövegrészletet kap, és egy olyan RGB képet hoz létre belőle, amely illeszkedik a leíráshoz. Például ha a bemenet az *"ennek a virágnak sok kicsi, kerek rózsaszín szirma van"*, akkor egy kerek, rózsaszín szirmú virág képét generálja.

A szöveges leírást először *embedding* (numerikus vektor) alakra hozzuk, majd egy véletlen zajvektort konkatenálunk hozzá, és ez alkotja a generátor bemenetét. A diszkriminátor pedig a generált képet



9.21. ábra. *Text-to-Image GAN működése*

és a szöveg embeddinget kapja meg bemenetéül, és immár nemcsak azt kell eldöntenie, hogy a kép valódi-e vagy hamis, hanem azt is, hogy a kép illik-e a szöveghez. (A diszkriminátor kimenete továbbra is 0 vagy 1 lehet, csak a döntési folyamat lett komplexebb.)

További Olvasnivaló

[1] Jeff Heaton. „Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning". *Genetic Programming and Evolvable Machines* 19.1-2 (2017. okt.), 305–307. old. DOI: 10.1007/s10710-017-9314-z. URL: https://doi.org/10.1007%2Fs10710-017-9314-z.

[38] Ian J. Goodfellow, Jonathon Shlens és Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2015. eprint: 1412.6572. URL: http://www.arxiv.org/abs/1412.6572.

[39] Leon A. Gatys, Alexander S. Ecker és Matthias Bethge. *A Neural Algorithm of Artistic Style*. 2015. eprint: 1508.06576. URL: http://www.arxiv.org/abs/1508.06576.

[40] Yongcheng Jing és tsai. *Neural Style Transfer: A Review*. 2018. eprint: 1705.04058. URL: http://www.arxiv.org/abs/1705.04058.

[41] Diederik P Kingma és Max Welling. *Auto-Encoding Variational Bayes*. 2014. eprint: 1312.6114. URL: http://www.arxiv.org/abs/1312.6114.

[42] Ian J. Goodfellow és tsai. *Generative Adversarial Networks*. 2014. eprint: 1406.2661. URL: http://www.arxiv.org/abs/1406.2661.

[43] Alec Radford, Luke Metz és Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. eprint: 1511.06434. URL: http://www.arxiv.org/abs/1511.06434.

[44] Mehdi Mirza és Simon Osindero. *Conditional Generative Adversarial Nets*. 2014. eprint: 1411.1784. URL: http://arxiv.org/abs/1411.1784.

[45] Xi Chen és tsai. *InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets*. 2016. eprint: 1606.03657. URL: http://www.arxiv.org/abs/1606.03657.

[46] David Berthelot, Tom Schumm és Luke Metz. *BEGAN: Boundary Equilibrium Generative Adversarial Networks*. 2017. eprint: 1703.10717. URL: http://arxiv.org/abs/1703.10717.

[47] Tero Karras és tsai. *Progressive Growing of GANs for Improved Quality, Stability, and Variation*. 2017. eprint: 1710.10196. URL: http://arxiv.org/abs/1710.10196.

[48] Martin Arjovsky, Soumith Chintala és Léon Bottou. *Wasserstein GAN*. 2017. eprint: 1701.07875. URL: http://arxiv.org/abs/1701.07875.

[49] Jun-Yan Zhu és tsai. *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*. 2017. eprint: 1703.10593. URL: http://arxiv.org/abs/1703.10593.

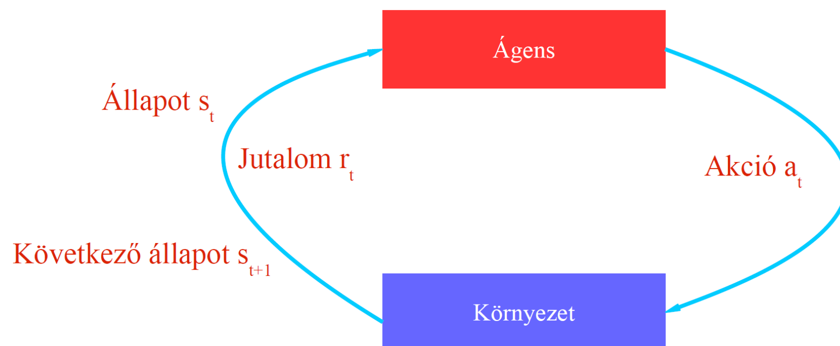
[50] Kai Hu és tsai. *Text to Image Generation with Semantic-Spatial Aware GAN*. 2021. eprint: 2104.00567. URL: http://arxiv.org/abs/2104.00567.

10. fejezet

Megerősítéses Tanulás

A korábbi előadásokon megismerkedtünk a felügyelt tanulás módszereivel. A felügyelt tanulásnak azonban számos hátránya van: Egyrészt a legtöbb esetben nagy méretű címkézett adatbázis szükséges hozzá, aminek előállítása költséges. Másrészt ezekkel a módszerekkel tulajdonképpen csak a saját képességeinket ”másoljuk” át a mesterséges intelligencia modellbe, de nem magában az algoritmusban fejlődik ki ez magától. A felügyelt tanulás elméleti lehetőségeit tehát a saját limitációink korlátozzák.

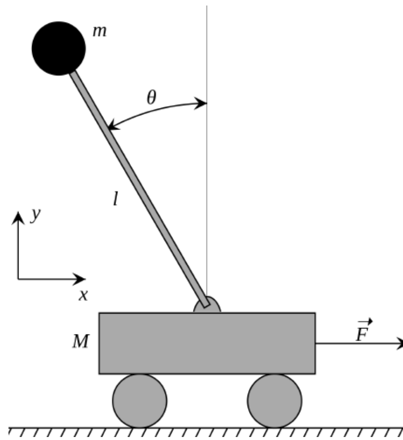
A megerősítéses tanulás során azonban az algoritmus (ágens) egy külső környezettel lép interakcióba, a célja pedig valamilyen feladat önálló megoldása ebben a környezetben. A probléma megfogalmazható úgy, hogy a környezetnek az ágens által megfigyelhető állapotai vannak, az ágens ez alapján dönt az általa végrehajtandó akcióról, majd ennek hatására a környezet egy új állapotba lép. Fontos eleme a megerősítéses tanulás környezetének a jutalom, a környezet által előállított és megfigyelhető visszajelzés, és az elvégzendő feladat teljesítésének minőségével áll összefüggésben. Ez a jutalom a legtöbb esetben csak ritkán áll elő, nem pedig minden időpillanatban.



10.1. ábra. *Az ágens és környezet interakciója.*

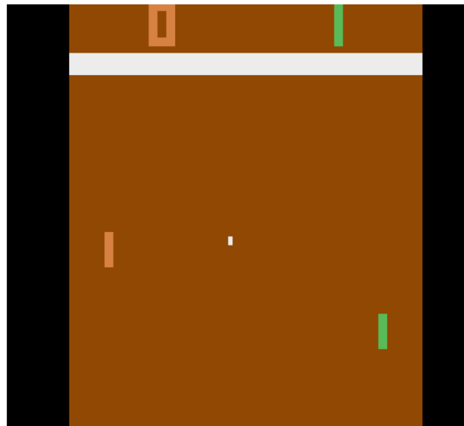
Fontos megjegyezni, hogy a felügyelt tanulással ellentétben a helyes, vagy legjobb akcióról nincs információnk, ráadásul, ha a jutalom csak a relatíve hosszú döntéssorozat végén áll elő, akkor azt sem tudjuk, hogy a mely akciók milyen irányban befolyásolták ezt a végső jutalmat. Ilyen környezet esetén a hagyományos tanulási paradigma nem alkalmazható.

Megerősítéses tanulásra számos példát lehet mondani, ezek közül az egyik legegyszerűbb az inverz inga (Cart-Pole problem) fenntartása, melynek során a környezet állapota a kocsi pozíciójából, sebességéből, valamint az inga kitérési szögéből és szögsebességéből áll. Akcióként értelmezhetjük a kocsira kiadandó oldalirányú erőt, a jutalom pedig minden időpillanatban 1, amíg az inga nem tér ki a függőleges pozíciójából túlságosan. Hasonlóan megfogalmazhatók számítógépes játékok is megerősítéses tanulási probléma formájában, ahol az állapotot a képernyő pixelei adják meg, az akciók terét pedig a játék irányítására használt gombok/eszközök. A jutalmat itt a játék megnyerése, vagy az abban elérhető pontok száma jelenti.



10.2. ábra. *Az inverz inga probléma.*

Természetesen számos más probléma is megfogalmazható ilyen formában, komplex robotok és járművek irányításától a különböző táblajátékokon (sakk, dáma, go) keresztül egészen magáig az életig.



10.3. ábra. *Egy mesterséges intelligencia (jobb), amely a számítógépes ágens (bal) ellen játszik a pong nevű Atari játékban.*

10.1. Q Learning

Első sorban ismerkedjünk meg a megerősítéses tanulás egyik klasszikus módszerével, a Q-Tanulással.

10.1.1. Markov döntési folyamat

A megerősítéses tanulás módszereinek bevezetéséhez először szükségünk van a probléma matematikai leírásának megadására. A megerősítéses tanulást egy úgynevezett Markov döntési folyamatként (MDP) szokás megadni. Természetesen a HMM (rejtett Markov modell) módjára az MDP is teljesíti a Markov tulajdonságot, vagyis az aktuális állapot teljes mértében leírja a világot, és annak teljes múltját. Egy MDP az alábbi struktúrával írható le:

$$(S, A, R, P, \gamma) \tag{10.1}$$

Ahol S az állapotok halmaza, A az akciók halmaza, $R(s, a)$ a jutalmak eloszlása egy adott állapot-akció párhoz, míg $P(s, a)$ a következő állapot eloszlása. γ a diszkont ráta, amivel a korábbi jutalmakat exponenciálisan súlyozzuk. Ennek értelme, hogy a korábbi akciók hatását az aktuális

jutalomra folyamatosan csökkentsük. Az MDP által leírt környezetben létező ágens legfőbb tulajdonsága a stratégia π (policy), amely egy olyan függvény, ami minden állapothoz hozzárendel egy akciót (vagy azok egy valószínűségi eloszlását). A megerősítéses tanulás célja, hogy találjuk meg azt a stratégiát, ami maximalizálja a teljes jutalom várható értékét.

10.1.2. Érték és Q függvény

A Q-Learning egyik fontos alapfogalma az érték függvény, amely egy adott stratégia követése esetén definiálható. Amennyiben ezt a stratégiát követjük, kapunk egy állapot-akció-jutalom hármasokból álló útvonalat, trajektóriát. Ennek alapján tudjuk definiálni az érték függvényt, ami azt adja meg, hogy ennek a stratégiának a követése mellett egy adott s állapotból indulva mi a jövőbeli jutalom várható értéke, vagyis mennyire jó nekünk ez az állapot.

$$V^{\pi}(s) = E\Big[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, \pi\Big] \tag{10.2}$$

Fontos megjegyezni, hogy a várható érték azért van az érték függvény definíciójában, mert adott esetben sem az állapotátmenet, sem a stratégia nem feltétlenül determinisztikus. Az érték függvény mintájára definiálhatjuk a Q-függvényt, ami nem az állapothoz, hanem egy állapot-akció párhoz rendeli hozzá a jövőbeli várható jutalmat adott stratégia követése mellett.

$$Q^{\pi}(s,a) = E\Big[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a = a_0, \pi\Big] \tag{10.3}$$

Könnyű belátni, hogy a két függvény között az alábbi kapcsolat áll fenn:

$$V^{\pi}(s) = \sum_{a'} Q^{\pi}(s,a') p^{\pi}(a') \tag{10.4}$$

Ahol $p^{\pi}(a)$ az a akció meglépésének valószínűsége a π stratégia szerint. Ezen felül definiálhatjuk az optimális Q-függvényt, ami triviálisan az összes π stratégia közül a maximális Q függvény. Az optimális Q-függvény segítségével definiálható az optimális érték függvény is, ami egy adott állapotban definiált Q-függvények maximuma (hiszen egy adott állapotból elérhető legnagyobb jutalom triviálisan az, amit a legnagyobb jutalmat eredményező akció meglépésével elérhetünk). A lehető legjobb stratégiánk pedig triviálisan az optimális Q-függvény szerinti legjobb akció meglépése.

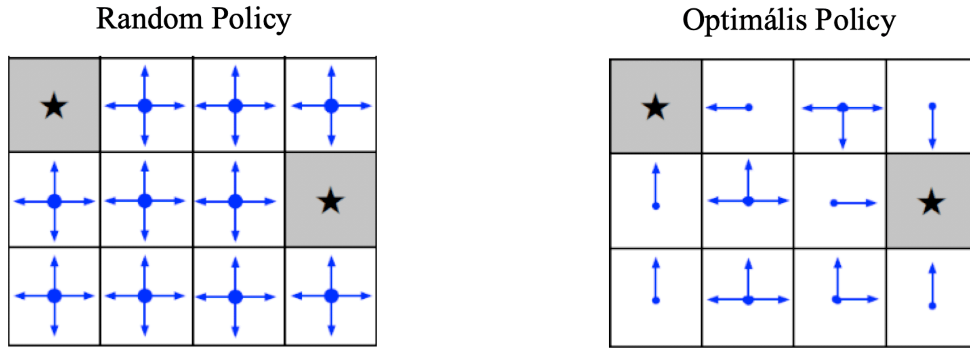
$$\begin{gathered} \hat{Q}(s,a) = max_{\pi} E\Big[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a = a_0, \pi\Big] \\ \hat{V}(s) = max_{a'} \hat{Q}(s,a') \end{gathered} \tag{10.5}$$

10.1.3. Bellman szabály

Az optimális Q-függvény egy fontos tulajdonsága, hogy teljesíti az úgynevezett Bellmann-egyenletet:

$$\hat{Q}(s,a) = E\Big[r + \hat{V}(s')\Big] = E\Big[r + max_{a'} \hat{Q}(s',a')\Big] \tag{10.6}$$

A Bellman-egyenlet szemléletesen azt jelenti, hogy egy adott állapot-akció párból a lehető legnagyobb jutalom megegyezik a közvetlenül kapott jutalom, és a következő állapotból elérhető legnagyobb jutalom összegével. Könnyen belátható, hogy optimális stratégia esetén ennek igaznak kéne lenni, hiszen, ha lehetne ennél nagyobb várható jutalmat elérni, akkor nem lenne optimális a



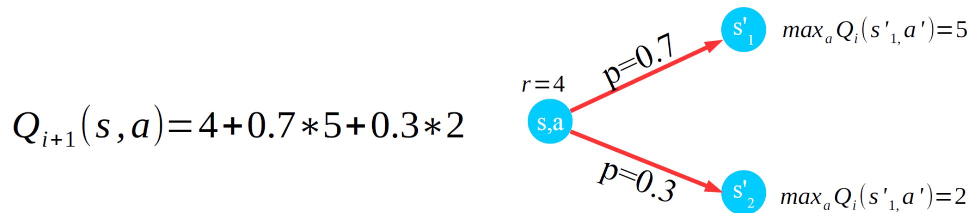
10.4. ábra. *A Random és az optimális stratégia a csillaggal megjelölt területek egyikébe való eljutásra.*

stratégia. Innen következik, hogy a Bellman egyenlet csak és kizárólag az optimális stratégiához tartozó Q-függvényre igaz, különben ellentmondásra jutunk.

Ebből született az öltet, hogy a Bellman-egyenlet felhasználható, mint egy iteratív algoritmus update-szabálya ahhoz, hogy egy véletlenül inicializált Q függvényből előállíthassuk az optimális Q függvényt, amiből az optimális stratégia már triviálisan meghatározható. Ez az érték iteráció algoritmusa, aminek a lépésszabálya a következő:

$$\hat{Q}_{i+1}(s,a) = E\Big[r + max_{a'}\hat{Q}_i(s',a')\Big] \tag{10.7}$$

Az értékiteráció algoritmusa során az algoritmus a környezettel történő interakciók során minden lépésben frissíteni a Q függvény éppen aktuális állapot-akció párhoz tartozó értékét, amíg az az optimális értékhez nem konvergál. Ennek a megoldásnak az egyik legnagyobb problémája, hogy a környezetnek adott esetben akár több ezer, vagy millió állapota is lehet, így az egész Q-függvény megtanulása rendkívül nehézzé válhat.



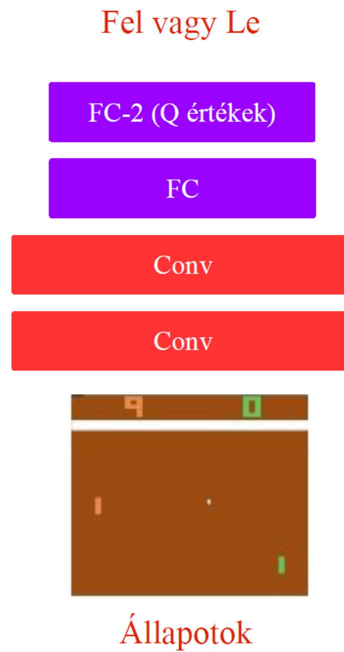
10.5. ábra. *A Bellman-szabály illusztrálása két lehetséges következő állapot esetén.*

10.1.4. DQN

Ennek a problémának a kezelésére alkalmazhatunk neurális hálókat. Elképzelhető ugyanis, hogy a Q-függvény aránylag sima, így a számos értéke ellenére egy mély neurális háló segítségével mégis jól közelíthető. ezt a megoldást hívjuk Mély Q-Hálónak (DQN). A neurális háló feladata tehát, hogy az állapotot és az akciót bemenetként megkapva a Q függvény értékét előállítsa. Ehhez nincs más dolgunk, mint a Bellman-egyenlet hibáját felírni, és ennek a négyzetét használni költségfüggvényként. Így a költségfüggvény és a deriváltja a következőképp adódnak:

$$\begin{gathered}\hat{Q}(s,a,\Theta) = E\Big[r + max_{a'}\hat{Q}(s',a',\Theta)\Big]\\ \Delta Q = r + max_{a'}Q(s',a',\Theta_{i-1}) - Q(s,a,\Theta_i)\\ L_i(\Theta_i) = E[\Delta Q_i^2]\\ \frac{\partial L_i(\Theta_i)}{\partial \Theta_i} = E[\Delta Q \frac{\partial Q(s,a,\Theta_i)}{\partial \Theta_i}]\end{gathered} \tag{10.8}$$

Ahol Θ a háló paraméterei, a Q függvény Θ szerinti deriváltja pedig a backpropagation segítségével számolható. Érdemes megjegyezni, hogy a hiba kifejezésében az egyik tagban azért használjuk a Θ korábbi értékét, hogy az a gradiensben ne szerepeljen, hiszen ez a hagyományos felügyelt regressziós felfogásban a tanító adat által előírt kimenetnek felelne meg.



10.6. ábra. *Egy DQN háló architektúra a pong játék esetére.*

A DQN tanítás egyik fontos gyakorlati praktikája a tapasztalat visszajátszás. E mögött az a motiváció rejlik meg, hogy mivel a Q függvény értéke meghatározza a következő akciót, ami pedig a következő állapotot, ezért az egymás utáni mintákkal tanított DQN be tudja magát zárni lokális hurkokba, ahonnan aztán sosem jut ki. Éppen ezért célszerű a hálót véletlen átmenetekből álló minibatch-ekkel tanítani. Ehhez a háló korábbi tapasztalatait egy tömbbe elmentjünk, majd ezt, mint egy tanító adatbázist felhasználva tanítjuk a hálót.

Bár a DQN a hagyományos Q-tanulásnál lényegesen nagyobb sikerrel működik, még mindig problémája az, hogy a Q függvény adott esetben borzalmasan komplex tud lenni, így nagy hálóra van szükség ennek megtanulására. Ebben az esetben a DQN módszer is csak limitált eredményeket képes hozni.

10.2. Policy Gradiens (REINFORCE)

Ennek ellenére gyakorta előfordul, hogy a rendkívül bonyolult Q függvény ellenére maga a stratégia relatíve egyszerű. Így adja magát a kérdés, hogy vajon lehetséges-e közvetlenül megtanulni. Egy ilyen neurális háló meglehetősen hasonlítana a hagyományos osztályozó neurális hálókra (legalábbis diszkrét akciók esetén), ugyanis a cél az lenne, hogy egy adott állapot esetén rendeljen valószínűségeket az egyes akciókhoz, ami alapján választani tudunk.

A tanítás során egyszerűen a jövőbeli diszkontált jutalom várható értékét kell maximalizálnunk, feltéve, hogy a háló által megvalósított stratégiát követjük. Erre alkalmazhatjuk a gradiens alapú maximalizálás módszerét.

10.2.1. Gradiens meghatározása

Ehhez nincs más dolgunk, mint a költségfüggvényt lederiválni a háló paraméterei szerint. Ehhez először kifejtjük a költségfüggvényt az alábbi módon:

$$J(\Theta) = E[r(\tau)] = \int_\tau r(\tau)p(\tau;\Theta) \tag{10.9}$$

Ahol τ a háló által implementált stratégia következtében kialakuló trajektória. Ennek deriváltja Θ szerint:

$$\nabla_\Theta J(\Theta) = \int_\tau r(\tau)\nabla_\Theta p(\tau;\Theta) \tag{10.10}$$

Itt azonban egy nagy problémába ütközünk: ez az integrál megoldhatatlan, még diszkrét esetben is, amikor szummára redukálódna. Ezen azonban tudunk valamelyest javítani az egyenlet módosításával. A deriválás azonosságát felhasználva felírhatjuk, hogy:

$$\nabla_\Theta p(\tau;\Theta) = p(\tau;\Theta)\frac{\nabla_\Theta p(\tau;\Theta)}{p(\tau;\Theta)} = p(\tau;\Theta)\ \nabla_\Theta log\ p(\tau;\Theta) \tag{10.11}$$

Ezt az eredeti kifejezésbe visszahelyettesítve:

$$\nabla_\Theta J(\Theta) = \int_\tau r(\tau)\nabla_\Theta log\ p(\tau;\Theta)\ p(\tau;\Theta) = E[r(\tau)\ \nabla_\Theta log\ p(\tau;\Theta)] \tag{10.12}$$

Érthető módon gondolhatjuk, hogy ”Na, hát most már tényleg sokkal szebb lett a kifejezés”. Azonban láthatjuk, hogy a hiba gradiense előáll a jutalom és a trajektória valószínűségének deriváltjának várható értékét vesszük. Ha a szorzat utolsó tagját meg tudjuk határozni, akkor ez a várható érték Monte-Carlo módszerrel becsülhető. A Monte-Carlo módszer lényege az, hogy véletlen mintavételezés segítségével a várható értéket az átlaggal közelítjük. Gondoljunk bele, hogy a sztochasztikus gradiens módszer alkalmazása esetén már eddig is ezt csináltuk minibacthenként.

Kérdés azonban, hogy a trajektória valószínűségének gradienseit meg tudjuk-e határozni anélkül, hogy az állapotátmenetek valószínűségét ismernénk. Könnyen fölírható ugyanis, hogy a trajektória valószínűsége:

$$p(\tau;\Theta) = \prod_{t\geq 0} p(s_{t+1}|s_t,a_t)\ \pi_\Theta(a_t|s_t) \tag{10.13}$$

Amihez szükségünk van a tranzíciókra. Szerencsére azonban ennek először a logaritmusát kell venni:

$$log\ p(\tau;\Theta) = \sum_{t\geq 0} log\ p(s_{t+1}|s_t,a_t) + log\ \pi_\Theta(a_t|s_t) \tag{10.14}$$

Ezt követően, ha ezt a háló paraméterei szerint deriváljuk, akkor az állapotátmenetek valószínűsége kiesik, hiszen ezek nem függnek a háló paramétereitől.

$$\nabla_\Theta log\ p(\tau;\Theta) = \sum_{t\geq 0} \nabla_\Theta log\ \pi_\Theta(a_t|s_t) \tag{10.15}$$

Végeredményben tehát ezt a képletet kapjuk a költségfüggvény deriváltjára:

$$\nabla_\Theta J(\Theta) = \sum_{t\geq 0} r(\tau)\nabla_\Theta log\ \pi_\Theta(a_t|s_t) \tag{10.16}$$

Vagyis a derivált a háló kimenetének logaritmusának deriváltjával arányos (ebben az esetben a jutalommal súlyozva). Vegyük észre, hogy ez valamelyest hasonlít az osztályozás során ismert

kereszt-entrópia függvény során kapott hibára, ahol szintén a helyes osztály valószínűségének negatív logaritmusát kellett vennünk. A különbség, hogy ebben az esetben ezt a helyességet a jutalom értéke határozza meg.

Az így kapott módszert Policy Gradiens, vagy egyes esetekben REINFORCE algoritmusnak nevezzük. Bár a levezetése meglehetősen bonyolult, az algoritmus meglehetősen könnyen értelmezhető: A háló végrehajt egy sor akciót, amiért valamilyen jutalmat kap. Ezt követően, ha a jutalom jó, akkor a hálót úgy módosítjuk, hogy a következő hasonló esetben nagyobb valószínűséggel hajtsa végre ugyanezt az akciósorozatot, vagyis megerősítjük a hálót a döntésében. Ellenkező esetben úgy módosítjuk a hálót, hogy ugyanannak az akciósorozatnak a végrehajtása kisebb eséllyel történjen meg, vagyis ellenezzük a döntést.

10.2.2. Zajcsökkentés és baseline

A Policy Gradiens módszernek (és a megerősítéses tanulásnak általában) az egyik központi problémája az úgynevezett credit-assignment probléma. Ez azt jelenti, hogy a döntéssorozat végrehajtása esetén nem tudjuk megmondani, hogy melyik akció volt a jutalomért igazán felelős. Ennek a problémának a következtében a gradiens becslése rendkívül zajos lehet, ami megnehezítheti, vagy meghiúsíthatja a konvergenciát.

Ennek kezelésére több módszert is szokás együttesen használni. Egyrészt a költséget úgy módosítjuk, hogy ne a teljes trajektória jutalmát vegye egy adott akció esetén figyelembe, hanem csak a konkrét akció után kapottakat. Ezen felül gyakran diszkontáljuk a jutalmakat, hogy az időben túl távoli jutalmak kevésbé számítsanak az adott akció jóságába.

$$\nabla_\Theta J(\Theta) = \sum_{t\geq 0} \Big(\sum_{t'\geq t} \gamma^{t'-t} r_{t'} \Big) \nabla_\Theta log\ \pi_\Theta(a_t|s_t) \tag{10.17}$$

10.2.3. Actor-critic hálók

A Policy Gradiens módszer egy további problémája, hogy a környezetből érkező jutalom értékek a legtöbb esetben nemnegatív számok. Ebben az esetben minden esetben megerősítjük az algoritmus döntését, csak jobb stratégiák esetén éppenséggel nagyobb mértékben tesszük ezt. Ezt viszont a konvergencia szempontjából nem ideális, hiszen így a rossz lépéseket is bátorítani fogjuk csekély mértékben. Éppen ezért célszerű lenne kiszámolni egy alap (baseline) jutalom értéket, és az aktuális jutalomból ezt az értéket levonni. Így a baseline-nál jobb teljesítményt jutalmazzuk, a rosszabbat pedig büntetjük. A baseline jutalom számításának legegyszerűbb módja a korábbi jutalmak mozgóátlagának használata. Ezzel elérjük, hogy akkor jutalmazunk, ha az algoritmus a korábbi átlagos teljesítményét képes felülmúlni.

$$\nabla_\Theta J(\Theta) = \sum_{t\geq 0} \Big(\sum_{t'\geq t} \gamma^{t'-t} r_{t'} - b(s_t) \Big) \nabla_\Theta log\ \pi_\Theta(a_t|s_t) \tag{10.18}$$

Egy másik - szofisztikáltabb - megoldás, ha akkor tekintünk egy jutalmat jónak, ha az nagyobb, mint az adott állapotból elérhető jutalom várható értéke. Ha ez a megfogalmazás ismerősen hangzik, akkor ez nem véletlen: ez pontosan a korábban megismert érték függvény definíciója. Tulajdonképpen definiálhatjuk az adott akció-állapot párhoz tartozó előnyfüggvényt, amely az állapot-akció párhoz tartozó Q függvény és az állapothoz tartozó érték függvény különbsége.

$$\begin{gathered} \nabla_\Theta J(\Theta) = \sum_{t\geq 0} \Big(Q^{\pi_\Theta}(s_t, a_t) - V^{\pi_\Theta}(s_t) \Big) \nabla_\Theta log\ \pi_\Theta(a_t|s_t) \\ A^{\pi_\Theta}(s,a) = Q^{\pi_\Theta}(s,a) - V^{\pi_\Theta}(s) \end{gathered} \tag{10.19}$$

E mögött az a ráció, hogy ha a stratégia (következésképp a Q és érték függvények) nem optimálisak (márpedig a tanítás közben nem azok), akkor van olyan akció, amelyik egy adott állapotból jobb, mint az állapot érték függvénye. Ha ilyen akciót léptünk, akkor jutalmazunk, ha nem akkor büntetünk. A fent leírt előny függvényt természetesen a korábban megismert DQM módszerrel előállíthatjuk.

Az ilyen megoldást használó eljárást Actor-Critic módszernek hívjuk. Ezek a fent leírtak alapján két külön hálóból állnak: az Actor háló a Policy Gradiens módszerrel tanulja az optimális stratégiát, míg a Critic háló pedig Q-tanulás segítségével az előnyfüggvényt igyekszik előállítani. Érdemes megjegyezni, hogy ennél a módszernél is érdemes a korábban ismertetett tapasztalat visszajátszás elvét használni.

Az Actor-Critic hálók meglehetősen jó minőségű működést képesek elérni a megerősítéses tanulásos feladatok esetében. Érdemes azonban megjegyezni, hogy a módszernek számos további változata is létezik. Ezek közül az egyik legfőbb, az úgynevezett Asynchronous Advantage Actor-Critic (A3C) modell, amely egyszerre több ágenst futtat külön környezetekben párhuzamosan. A modell paraméterek ennek megfelelően sztochasztikus módon kerülnek frissítésre, így nincs szüksét a tapasztalat visszajátszásra. Ennek a módszernek egy másik változata az A2C, amely nem aszinkron módon végzi a paraméter frissítéseket, így nem fordul elő, hogy egyes modellek még a korábbi paramétereknek megfelelően működnek. A kísérletek tanúsága szerint az A2C modell jobb hatékonyságot tud elérni.

Érdemes továbbá még megemlíteni azt az esetet, amikor a policy háló kiemete nem véges számú akciók valamelyike, hanem egy folytonos mennyiség (vagyis a probléma regresszió jellegű). Ez gyakorta fordul elő a robotikában, ahol az akciók általában az egyes motorok nyomatékai. Ebben az esetben a policy háló kimenete nem az egyes akciók velószínűsége, hanem közvetlenül az akció értéke:

$$a_t = \mu_\Theta(s_t) \tag{10.20}$$

Ebben az esetben azonban egy nem várt problémába ütközünk: mi legyen a háló költségfüggvény gradiense? Korábban ez a probléma egyszerű volt, hiszen jó jutalom esetén növelni, rossz esetén meg csökkenteni szeretnénk az aktuális akció valószínűségét, vagyis a helyes akció ismerete nélkül is meg tudjuk mondani, hogy a kimenet merre változzon. A mostani példa során viszont tudnunk kellene, hogy a jutalom a kimenet milyen irányú változása esetén fog nőni. Ennek megoldására az Acotr-Critic módszerekhez hasonlóan felhaszálunk egy neurális hálót a Q függvény megbecslésére: ennek a bemenete az aktuális állapot és akció, kimenete pedig a jutalom várható értéke. Ennek segítségével a gradienst az alábbi módon definiálhatjuk:

$$\nabla_\Theta J(\Theta) = \sum_{t \geq 0} \nabla_{a_t} Q^{\pi_\Theta}(s_t, a_t) \nabla_\Theta log\ \mu_\Theta(s_t) \tag{10.21}$$

Vagyis a Backpropagation segítségével előállítjuk a Q-függvény akció szerinti deriváltját, ami pont azt mondja meg, hogy merre kellene az akciónak változni ahhoz, hogy a jövőbeli jutalom várható értéke növekedjen. Ezt az algoritmust nevezzük Deep Deterministic Policy Gradient (DDPG) módszernek, melynek szintén létezik több ágens együttes futtatásával kibővített (distributed) változata, a D4PG. Érdemes azt is megjegyezni, hogy a determinisztikus működés rossz felfedezőképességének elkerülése végett a poily háló kimenetét tanítás közben gyakorta terhelik véletlen zajjal.

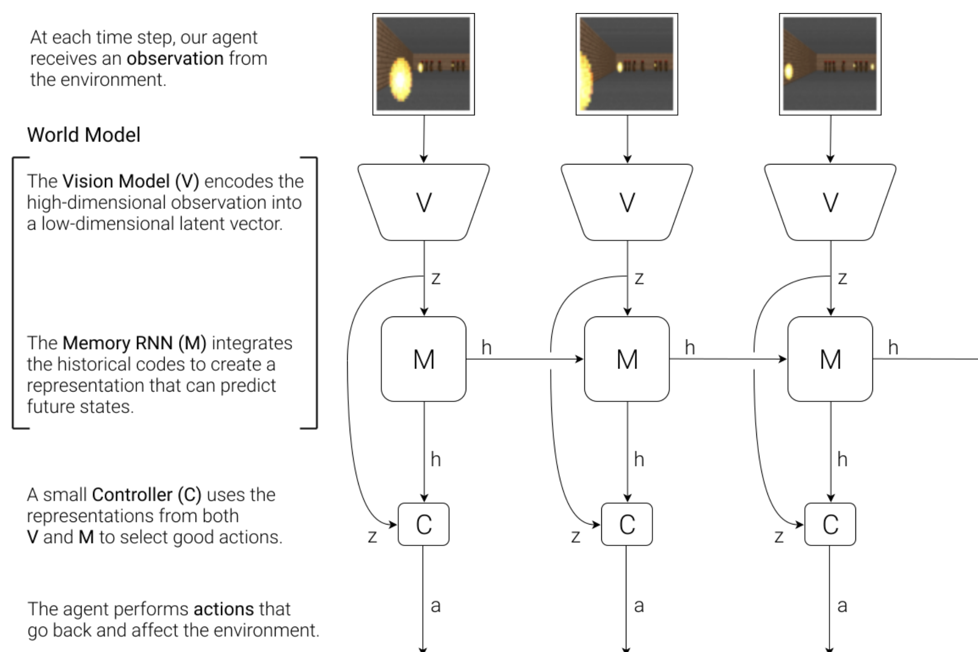
10.2.4. Ritka jutalom és kíváncsiág

Megerősítéses tanulás során akad jó néhány olyan eset, amikor a tanulás nehéz, vagy akár kvázi lehetetlen is számukra. Ezek közül az egyik eset az, amikor nagyon ritkán, csak komplex akciósorozatok hatására lehet jutalmat szerezni az adott környezetben. Mivel ezek a hálók véletlenszerűen cselekednek a tanulás elején, ezért statisztikailag rendkívül kicsi az esélye annak, hogy egy jó megoldásra véletlenül ráhibázzanak. Ezen a problémán lehet valamilyen mértékben segíteni azzal, hogy

a policy háló kimenetén adódó eloszlásból nem mindig a legvalószínűbb akciót választjuk, hanem az eloszlás alapján véletlenül mintavételezünk.

Ritkán előforduló külső jutalmak esetében egy rendkívül újszerű és intenzíven kutatott megoldás a különféle belső (intrinsic) jutalmakra alapulú tanuló rendszerek alkalmazása. Ez a belső jutalom a legtöbb esetben a kíváncsiság és a predikció fogalmával áll kapcsolatban. A modellek lényege, hogy az előző állapot és akció felhasználásával legyünk képesek a következő állapotot minél pontosabban megbecsülni, vagyis a környezet működését minél inkább megérteni. A gyakorlatban gyakra nem közvetlenül az állapot, hanem egy az állapotból generált belső leíró vektor következő értékét becsüljük. Ezt azért tesszük, mert az állapot gyakorta nagyon nagy dimenzionaliztású, és a benne szereplő információ nagy része nem releváns számunkra.

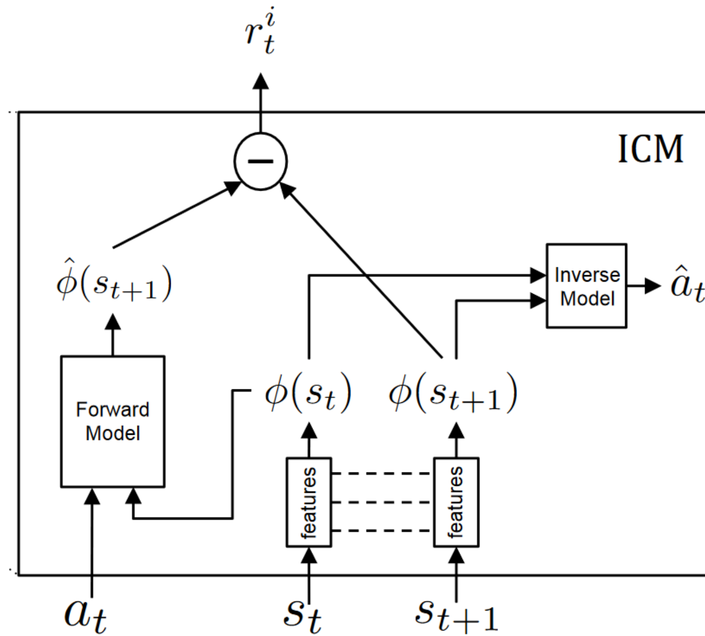
Az egyik ilyen megoldás az ún. világ modell (World Model) architektúra. Ez a struktúra 3 részből áll: A Vision modul azért felelős, hogy a háló által látott kép belső reprezentációját adja, a Memory modul pedig egy visszacsatolt cella, amely az egymást követő reprezentációk integrálásáért felelős. Az utolsó modul, a Controller, melynek feladata az akció meghatározása. A világmodell architektúra egyik fontos része, hogy a korábban megismert Autoencoderekhez hasonlóan, a belső reprezentációból a hálónak vissza kell tudnia állítani az eredeti állapotot.



10.7. ábra. *A világmodell felépítése.*

Az ilyen prediktív modelleket gyakran kombináljuk kíváncsiság mechanizmussal is, melynek során olyan állpotokba próbálunk eljutni, amelyekben még nem voltunk. Ezt gyakran a korábbi állapotok becsült eloszlása alapján mérjük: Az eloszlás szerint valószínűtlen állapotok újszerűnek, míg a valószínűek ismertnek számítanak. A környezet jobb megértésének elősegítésére gyakorta nem csak a következő állapotot, hanem két egymás követő állapot alapján az egyikből a másikba vezető akciót is becsüljük. Ez gyakran azért szükséges, mert a következő állapot egyszerűen becsülhető, ha a háló a csupa nulla belső reprezentációt rendeli minden állapothoz. Ekkor viszont nem lehetséges a két állapot közti akció visszafejtése.

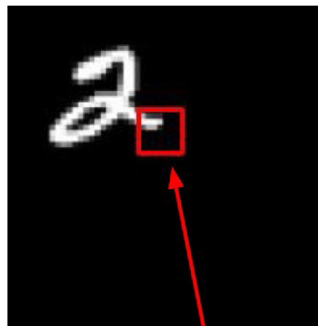
A megerősítésees tanulás során további nehézséget okoz, ha a környezetet leíró Markov döntési folyamatnak vannak rejtett állapotai. Ebben az esetben az algoritmusok konvergenciája kifejezetten nehézzé válik. Erre tipikusan jó példa a Starcraft (vagy hasonló real-time stratégiai játékok), ahol a széleskörben használt Fog of War mechanizmus miatt a játéktér jelentős része rejtve marad az algoritmus számára.



10.8. ábra. *A kíváncsiság modell felépítése.*

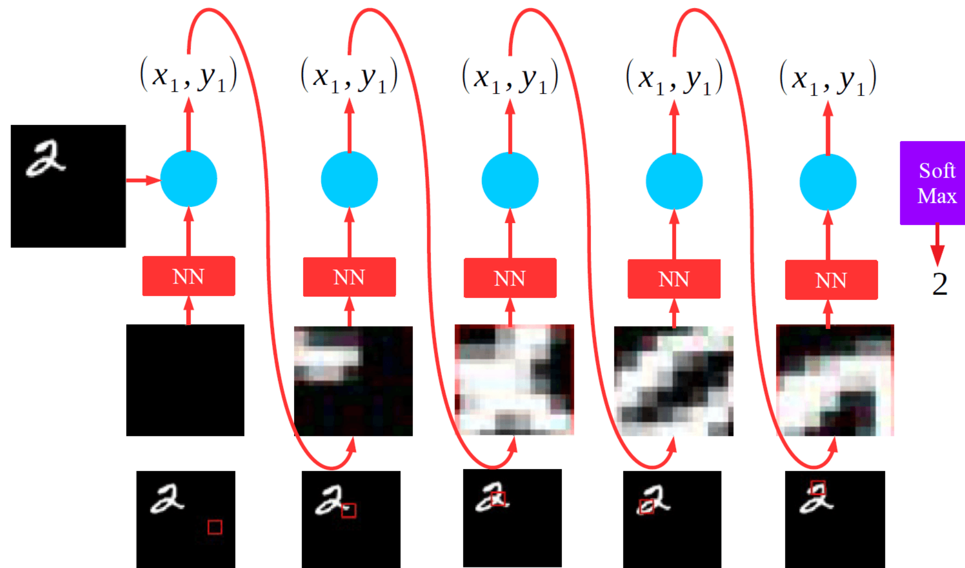
10.3. Kemény figyelem

Érdemes még megemlíteni a megerősítéses tanulás egy másik számítógépes látásban alkalmazott esetét, a kemény figyelmet. Korábban láthattuk, hogy az emberi (mentális) figyelemre hasonló mechanizmus létrehozható visszacsatolt neurális hálók segítségével. Ugyanez megtehető az emberi szemmozgás mintájára is: vagyis az algoritmus egy nagy méretű képet úgy dolgoz fel, hogy egymás után több kisebb képrészletet (”pillantást”) dolgoz fel. A feladat állapotai az eddig látott képrészletek lesznek, az akció pedig a következő pillantás koordinátái, jutalomként pedig +3-et kap az algoritmus, ha a végső osztályozás helyes.



10.9. ábra. *Egy pillantás.*

Érdemes megjegyezni, hogy maga a pillantás művelete nem deriválható, így ez hagyományos felügyelt tanulás segítségével nem megoldható. A Policy Gradiens módszere viszont lehetőséget ad arra, hogy a Monte-Carlo mintavételezéssel nem deriválható műveletek deriváltját numerikusan közelítsük, így a problémát meg tudjuk oldani.



10.10. ábra. *A kemény figyelem módszerét megvalósító háló architektúra.*

További Olvasnivaló

[1] Jeff Heaton. „Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning". *Genetic Programming and Evolvable Machines* 19.1-2 (2017. okt.), 305–307. old. DOI: 10.1007/s10710-017-9314-z. URL: https://doi.org/10.1007%2Fs10710-017-9314-z.

[51] Volodymyr Mnih és tsai. „Human-level control through deep reinforcement learning". *Nature* 518.7540 (2015. febr.), 529–533. old. DOI: 10.1038/nature14236. URL: https://doi.org/10.1038%2Fnature14236.

[52] Volodymyr Mnih és tsai. *Asynchronous Methods for Deep Reinforcement Learning*. 2016. eprint: 1602.01783. URL: http://www.arxiv.org/abs/1602.01783.

[53] Timothy P. Lillicrap és tsai. *Continuous control with deep reinforcement learning*. 2019. eprint: 1509.02971. URL: http://www.arxiv.org/abs/1509.02971.

[54] Yuhuai Wu és tsai. *Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation*. 2017. eprint: 1708.05144. URL: http://www.arxiv.org/abs/1708.05144.

[55] Deepak Pathak és tsai. *Curiosity-driven Exploration by Self-supervised Prediction*. 2017. eprint: 1705.05363. URL: http://www.arxiv.org/abs/1705.05363.

[56] David Ha és Jürgen Schmidhuber. *World Models*. 2018. eprint: 1803.10122. URL: http://www.arxiv.org/abs/1803.10122.

11. fejezet

Ön-felügyelt Tanulás

A korábbi fejezetekben megismerkedhedtünk a nem felügyet tanulás két különböző formájával, melyekben a képgenerálás, illetve egy környezetben történő feladatmegoldás problémáját kíséreltük megoldani. A GAN-ok tárgyalásakor kiemeltük, hogy a generátor által megtanult belső reprezentáció különösen kifejező, így akár hasznos lehet encoder neurális hálózatok előzetes tanítására. A GAN-ok tanítása viszont kifejezetten nehéz és körülményes, így felmerülhet a kérdés: Lehetséges-e egyszerűbb módon úgy előtanítani egy neurális hálózatot, hogy az későbbi feladatok megoldására jól használható jellemzőket tanuljon meg?

A válasz természetesen igen, és ennek módja a jelen fejezet témája. Ha visszaemlékszünk a korábban említett transzfer tanulás példájára, akkor észrevehetjük, hogy ott is egy nagyon hasonló problémát oldottunk meg, de az előtanításhoz egy másik - általában nagyobb - felcímkézett adatbázist alkalmaztunk. A jelen fejezetben olyan módszerekkel foglalkozunk, melyek egy címkézetlen képi adatbázison képesek tanulni úgy, hogy a feladatot a képekből automatikusan generálják, így a címkék is ugyanígy emberi munka nélkül előállíthatók. Ezt a megoldást nevezzük ön-felügyelt (self-supervised) tanulásnak.

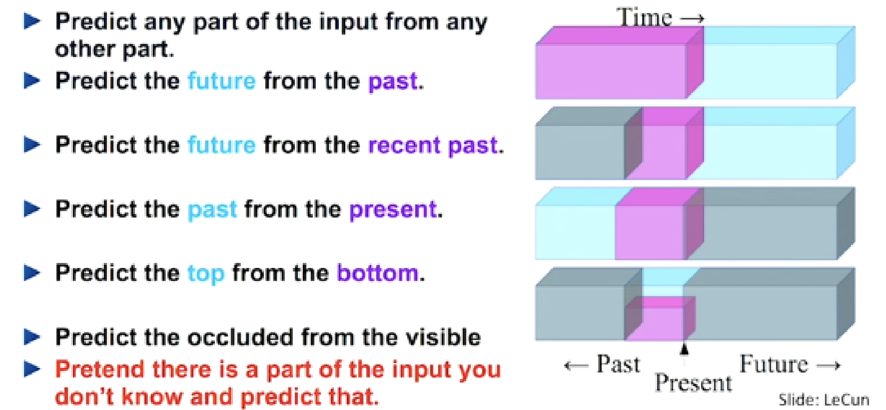
Az önfelügyelt tanulásnak több lehetséges sémája létezik: Az úgynevezett pre-text feladatok alkalmazása esetében explicit definiálunk egy feladatot az adott adatbázison, míg a kontrasztív tanulás esetében a bemeneti adatok közti hasonlóság alapján szeretnénk egy robusztus reprezentációt tanítani a hálózatnak. Célunk, hogy a folyamat végén egy olyan enkóder hálónk legyen, ami már rendkívül kicsi adatbázisok (akár osztályonként néhány kép) segítségével is hatékonyan tud tanulni.

11.1. Pre-text feladatok

Az önfelügyelt tanulás első fontos formája a pre-text feladatok megoldása. ezek olyan feladatok, melyek a tanításra használt képi adatbázisból automatikusan generálhatók, például az input egyik részét kimaszkolva előírjuk a hálónak, hogy a többi rész alapján rajzolja vissza a kimaszkolt területet minél pontosabban. Általánosságban, ha időben is kiterjedt inputot feltételezünk (pl.: videó), akkor generálhatunk feladatokat úgy, hogy a múlt-jelen-jövő hármasból bármelyikből bármelyik másikat megbecsültetjük a hálóval. Természetesen olyan is lehetséges, hogy a jelen információt több részre osztjuk és azt adjuk feladatnak, hogy az egyiket a másikból meg tudjuk becsülni. Mivel az input teljes egésze rendelkezésünkre áll, így ezek a feladatok automatikusan generálhatók.

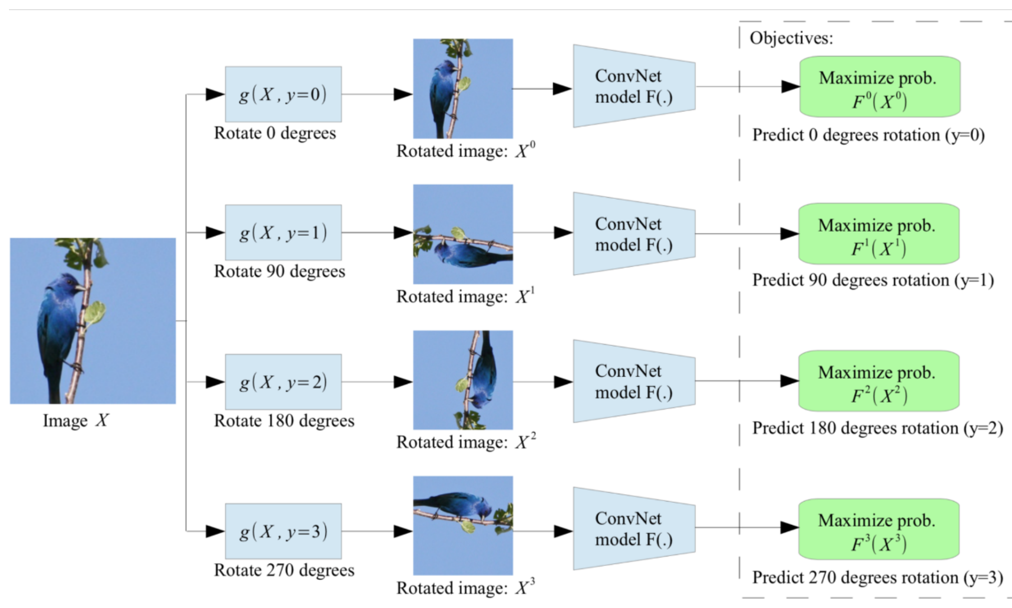
11.1.1. Kép alapú módszerek

Nézzük először azt az esetet, amikor csak állóképekből álló adatbázisunk van, hiszen ez a gyakorlatban lényegesen többször fordul elő. Az egyik legelső pre-text feladatra épülő módszer, a ROTATION, a képek forgatását aknázta ki. Az önfelügyelt módon tanítani kívánt konvolúciós enkóderre a kutatók egy egyszerű négy osztályt megkülönböztetni tudó osztályozó fejet tettek, majd



11.1. ábra. *Az ön-felügyelt tanulás elve.*

az adatbázisban lévő képeket véletlenszerűen elforgatták $0, 90, 180$, vagy 270 fokkal. Ezt természetesen ne előre csinálták meg, hanem felolvasás közben, így minden kép 25% eséllyel előfordult mind a négy lehetséges orientációban. Ezt követően a hálót arra tanították, hogy találja el, hogy az adott képek mennyivel voltak elforgatva (minden kimeneti osztály egy fajta elforgatásnak felelt meg).

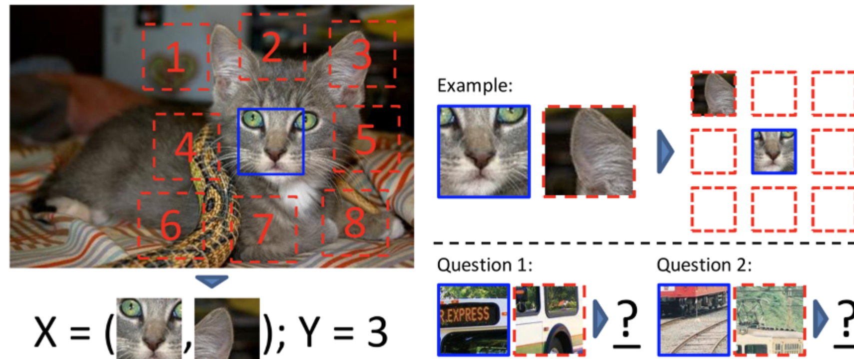


11.2. ábra. *A Rotation pre-text feladat megoldása.*

A következő érdekes módszer, az Exemplar-CNN, működése során az adatbázisban lévő képekből véletlenszerű pozíciókban és skálák mentén 32×32-es patch-eket vágott ki olyan helyekről, ahol a képi gradiensek megfelelően nagyon voltak (ezzel elkerülve a homogén régiók kivágását). Az így legyártott N darab patchen aztán különböző -meglehetősen durva - augmentációs eljárásokat végezve előállítunk mindegyikhez k darab torzított változatot. A hálózat feladata ezt követően, hogy megtanulja, hogy ugyanazon patch torzított változatait ugyanabba az osztályba sorolja be, így maga a feladat egy egyszerű osztályozás.

Egy következő érdekes módszer a Patches, melynek során kiválasztunk egy véletlenszerű patchet egy képről, majd körülötte egy 3×3-as griden még 8 másik patchet. Ennek a 8 másik patchnek a pozícióit enyhén randomizáljuk, hogy ne mindig pontosan ugyanonnan legyenek kivéve. Ezt követően a hálónak a középső és a többi patchet beadva egy osztályozást végzünk, melynek kimenetén elvárjuk, hogy a háló eltalálja, hogy melyik patch a középsőtől melyik irányban volt (itt a 8 lehetséges pozíció 1-1 osztály). Ennél a módszernél fontos megjegyezni, hogy nagyon fontos a patcheken

szín augmentációt végezni, ugyanis a valós kamerákkal készített képek esetében gyakori jelenség a kromatikus aberráció, amely a színcsatornákban egy pozíciófüggő torzulás. Ennek kihasználásával a háló könnyen meg tudná oldani a feladatot, anélkül, hogy a képi tartalom szempontjából hasznos reprezentációt tanulna meg.



11.3. ábra. *A Patches pre-text feladat megoldása.*

Egy további hasonló megoldás a Puzzle nevű pre-text feladat, melynek során a képből az előzőhöz hasonló módon kivágott 3×3-as griden lévő patcheket véletlenszerűen megpermutáljuk, a neurális hálózatnak pedig ki kell találni a helyes sorrendet. Itt érdemes megjegyezni, hogy ez a lehetséges permutációk nagy száma miatt neurális hálóknak egy kifejezetten nehéz feladat, így az eredeti megoldásban a hálónak csak 64 különböző előre definiált permutáció közül kellett egy osztályozással választani. Manapság azonban a transzformer hálók segítségével már lehetséges lehetne egy egyszerű architektúra segítségével mind a $9! = 362,880$ lehetséges permutációt figyelembe venni.

Végezetül érdemes megemlíteni a színezés (Colorization) nevű feladatot, ahol egy színes képet megfelelően megválasztott színtérbe (YCbCr, HSV, Lab, stb) konvertálunk, majd a hálózatnak a bementére adjuk a szürkeárnyalatos intezintásnak megfelelő csatornát, és elvárjuk, hogy a háló a kimeneten minden pixelhez becsülje meg a másik két színcsatorna értékét. Ehhez egy encoder-decoder architektúrát használunk, melyet a szemantikus szegmentálásnál, illetve az autoencodereknél is megismerhettünk. Érdemes megjegyezni, hogy az eredeti módszerrel a színeket nem regresszió, hanem osztályzás segítségével határozták meg úgy, hogy a színeket bin-ekre osztották, és ezek között kereszt-entrópia segítségével döntöttek.

11.1.2. Videó alapú módszerek

Természetesen amennyiben videó alapú input áll rendelkezésre, akkor is számos hasonló pre-text feladatot tudunk alkotni. Videók esetében leggyakrabban az időbeli konzisztenciát, illetve a mozgás működését igyekszünk az enkódernek megtanítani. Erre egy jó példa a Sequence feladat, ahol a háló bemenetére a videó képkockáit 50% valószínűséggel helytelen sorrendben adjuk, a hálónak pedig egy bináris osztályzással el kell döntenie, hogy helyes volt-e a sorrend. Hasonló feladat az Odd-One-Out, ahol a háló több helyes, és egy helytelen sorrendű szekvenciát kap, és a feladata, hogy helyesen azonosítsa a rossz sorozatot. Egy további módszer az Arrow of Time, ahol a háló a képsorozatot 50% eséllyel fordítva kapja meg, és el kell döntenie, hogy merre mutat az idő tengely.

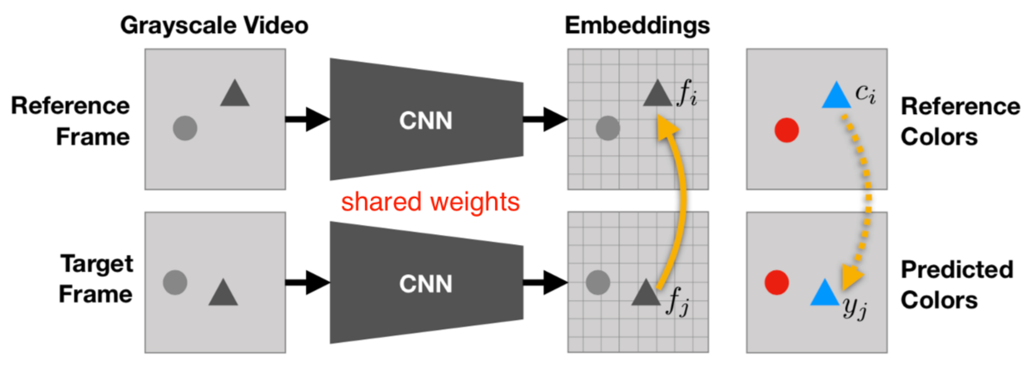
Egy különösen fontos módszer a videó színezés megoldása, ahol a videó két közeli képkockája átmegy ugyanazon a konvolúciós enkóderen, így minden pixelhez kapunk egy f_i leíró jellemzőt. Ezt követően az előző referencia kép minden i pixele és a target kép minden j pixele között egy figyelem értéket számolunk a már korábbról ismeretes SoftMax függvény segítségével:

$$A_{ij} = \frac{exp(f_i^T f_j)}{\sum_k exp(f_k^t f_j)}, \tag{11.1}$$

ahol f_k a referencia kép összes többi pixelének leírója. Ezt követően a target kép y_j pixelének értéke a következő módon adódik:

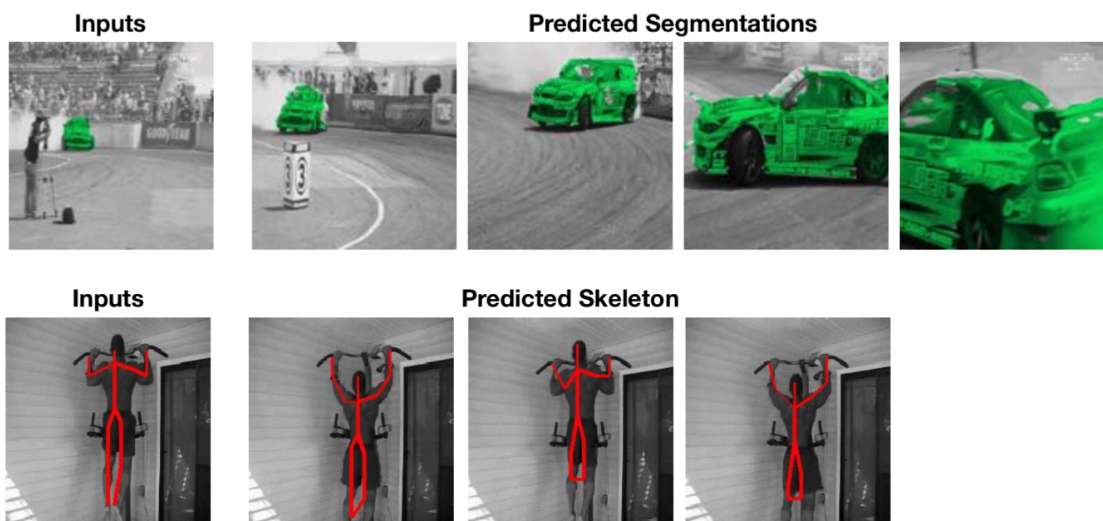
$$y_j = \sum_i A_{ij} c_i, \tag{11.2}$$

ahol c_i az i-edik referencia pixel színe. Tehát nincs másról szó, mint a kiszámolt figyelem értékek segítségével súlyozva átlagoljuk a referencia kép színeit. A módszert arra tanítjuk, hogy a következő képkockát ezzel a megoldással minél pontosabban legyen képes megbecsülni.



11.4. ábra. *A Video Colorization pre-text feladat megoldása.*

Ennek a megoldásnak létezik azonban egy kifejezetten hasznos alkalmazása az enkóder előtanításán felül is: Képzeljük el, hogy valamilyen módon (akár kézzel) a videó első képkockáján egy objektumot feljelölünk, és beszínezzük valamilyen a többitől jól elkülönülő színnel. Ezt követően, ha a videóra lefuttatjuk a fenti algoritmust, akkor ezeket a színeket feltételezhetően helyesen fogja a további képkockákra továbbterjeszteni. Így kaptunk egy olyan módszert, amely képes egy kezdeti szegmentáció alapján egy egész videót felcímkézni, vagy objektumokat helyesen követni.



11.5. ábra. *Követés videó színezés segítségével.*

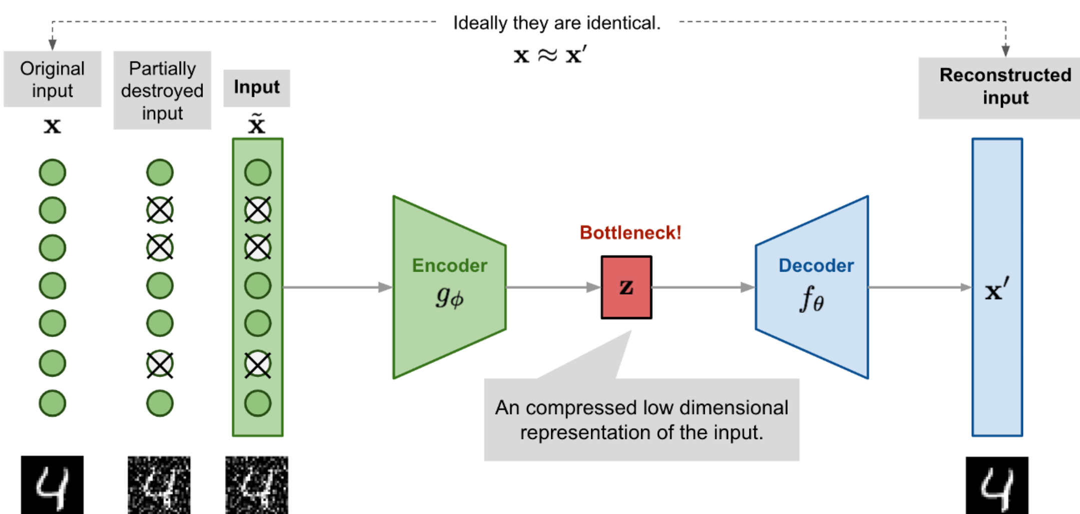
11.2. Generatív módszerek

A fent ismertetett pre-text feladatok egyik nagy előnye, hogy könnyen megvalósíthatók, és egyfajta vizuális "józan paraszti észt" tanítanak a neurális hálóknak, aminek eredményeképp feltehetően megnő az általánosítóképessége. Azonban gyakran ezeket a feladatokat kitalálni elég macerás, rengeteg lehetőségünk van, és nem világos, hogy melyik miben jobb a másiknál. Ráadásul az sem triviális, hogy ezek segítségével tényleg általános jellemzőket kapunk-e, ugyanis az encoder overfittelhet a pre-text feladatra is.

Éppen ezért érdemes valamelyest általánosabb pre-text feladatot kitalálni, ez pedig már a fejezet elején is megemlített predikció. Itt a lényeg az lesz, hogy a rendelkezésre álló input egyik részéből megpróbáljuk megbecsülni (vagyis minél pontosabban visszaállítani) a másikat. Ez egy lényegesen általánosabb feladat, azonban a megoldására a korábbi fejezetben megismert generatív módszerekre lesz szükségünk.

11.2.1. Autoencoderek

A generatív módszerek első része az Autoencoder architektúrákra alapszik. Ezek közül az első, és az egyik legfontosabb módszer az úgynevezett Denoising Autoencoder. Ez a megoldás nem csak itt, hanem a diffusion modellekben történő alkalmazás miatt is fontos. A denoising autoencoder - ahogy a neve is mutatja - egy szándékosan zajosított inputot kap, és a feladata, hogy az eredeti zajmentes inputot minél pontosabba előállítsa. Emiatt nem csak meg kell értenie, hogy mi egy képen a zaj és mi a hasznos jel, hanem ki is kell találnia a zaj miatt megsemmisült képrészleteken lévő információt a kontextusból.

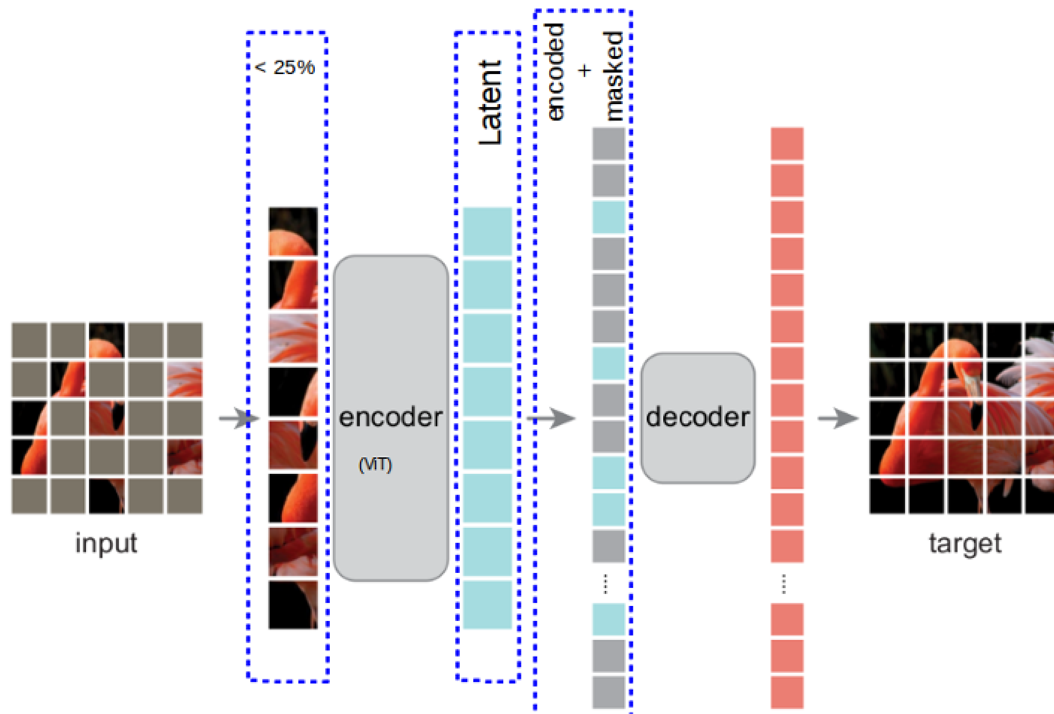


11.6. ábra. *A Denoising Autoencoder.*

A képszínezéshez hasonló módszer a Split-brain Autoencoder, melynek feladata, hogy a szürkeárnyalatos képből előállítsa a színcsatornát. Azonban ez a megoldás nem csak ezt végzi el, hanem ennek az inverzét is: a színcsatornából is ki kell találnia a szürkeárnyalatos komponenst, így gyakorlatilag egyszerre minden színkonverziós feladatot tudnia kell, emiatt feltehetően nő az általánosítóképessége.

Érdekes módszer még a Context Autoencoder, melynek feladata, hogy a kép közepéről kimaszkolt patch-et minél pontosabban visszaállítsa a kontextus alapján. Ennek egy lényegesen gyakrabban használt változata az úgynevezett Masked Autoencoder (MAE), mely nem egy középső patchet maszkol ki csupán, hanem a képet patchekre osztja, és ezek nagy többségét (általában több, mint 75%-át kimaszkolja. Az Autoencoder enkóder része csak a meghagyott patcheket kapja meg, és feladata, hogy a teljes inputot helyesen visszaállítsa.

Fontos eleme még ennek a módszernek, hogy a hibafüggvényben nem csak egy rekonstrukciós hibatag, hanem egy GAN hiba is helyet kapott. Ez azt jelenti, hogy a MAE-hez tartozik egy külön diszkriminátor háló, amit arra tanítanak, hogy tanulja meg megkülönböztetni az eredeti inputot a MAE által rekonstruálttól. A kutatások azt találták, hogy ez az extra tag lényegesen növeli a MAE teljesítményét. Összefoglalásképp megjegyzendő, hogy a MAE kifejezetten jó ön-felügyelt tanuló algoritmus, sőt még egyfajta adataugmentációnak/regularizációnak is használható más hálózatokban.



11.7. ábra. *A Masked Autoencoder.*

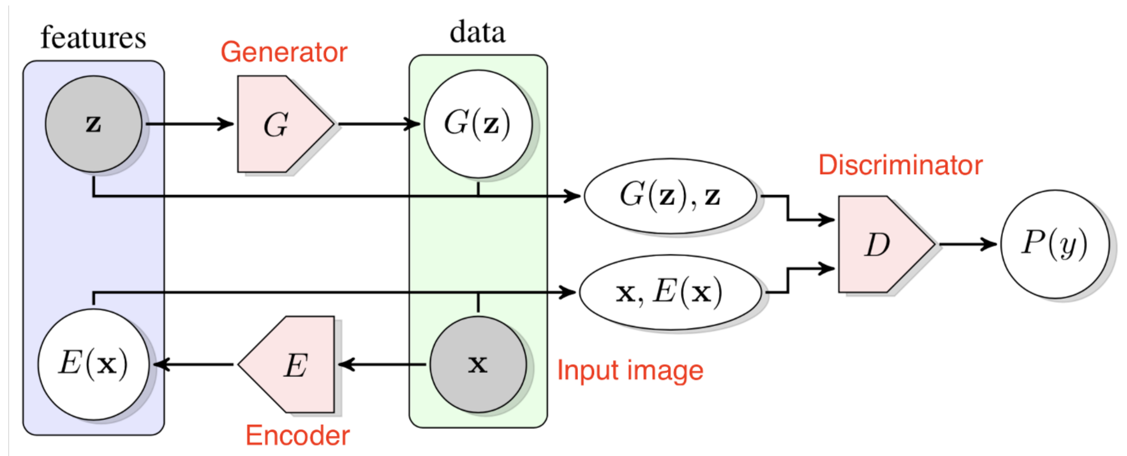
11.2.2. Bidirectional GAN

Mint már említettük, a GAN Generátor bemeneti vektora egy kifejezetten kifejező képleíró mennyiség, azonban ha ezt szeretnénk képek enkódolására használni, akkor abba a problémába ütközünk, hogy ehhez a Generátort invertálnunk kellene, ami nem triviális feladat (a Diszkriminátor ugyanis a Generátortól teljesen független jellemzőket tanul). Ennek a problémának a megoldására alkották meg az úgynevezett Bidirectional GAN módszert.

A Bidirectional GAN a hagyományos GAN-tól annyiban különbözik, hogy a Generátor és a Diszkriminátor mellett még egy Encoder háló is található, melynek célja, hogy az igazi képekből előállítson egy olyan jellemző vektort, ami akár a Generátor bemenete is lehetne. A Diszkriminátor annyiban módosul, hogy bemenetére nem csak a képet, hanem generált kép esetében a Generátor z bemenetét, valódi kép esetében pedig az Enkóder által generált $E(x)$ jellemző vektort is megkapja. Így az enkóder megtanul a képekhez rendkívül informatív jellemző vektorokat tanulni, tehát gyakorlatilag a Generátort invertálni.

11.3. Kontrasztív tanulás

Az eddigi módszerek vagy valamilyen explicit pre-text feladaton, vagy pedig valamilyen predikciós feladat elvégzésén alapultak. Az ötlet ezeknél az volt, hogy ezek elvégzéséhez a hálónak valamilyen



11.8. ábra. *A Bidirectional GAN.*

általánosan felhasználható képjellemzőket kell tanulnia, amik később más feladatokra is alkalmasak lettek. Már a pre-text feladatok tárgyalásánál is felvetettük azt, hogy ez nem feltétlenül igaz. Habár a különböző predikciók elvégzése általánosabb feladatnak tűnik, így is elgondolkozhatunk, hogy nem lehetséges-e valahogy a hálózatot explicit módon arra tanítani, hogy hasonló képeket az általuk megtanult reprezentációs téren belül hasonló helyre képezzenek le, míg különbözőket pedig eltérő helyekre.

A válasz az, hogy igen, lehet, és ezeket a módszereket összességében kontrasztív tanuló algoritmusoknak nevezzük. Formálisan egy ilyen algoritmus esetén létezik egy referencia példa x, illetve ahhoz viszonyítva pozitív példák x^+, illetve negatív példák x^- halmaza. A tanuló algoritmus egy f függvény, mely a bemeneti példákat leképezi valamilyen absztrakt jellemző térbe, és azt szeretnénk, hogy $f(x)$ és $f(x^+)$ legyen hasonló, míg $f(x)$ és $f(x^-)$ értékei legyenek különbözők, vagyis:

$$Score(f(x), f(x^+)) >> Score(f(x), f(x^-)) \tag{11.3}$$

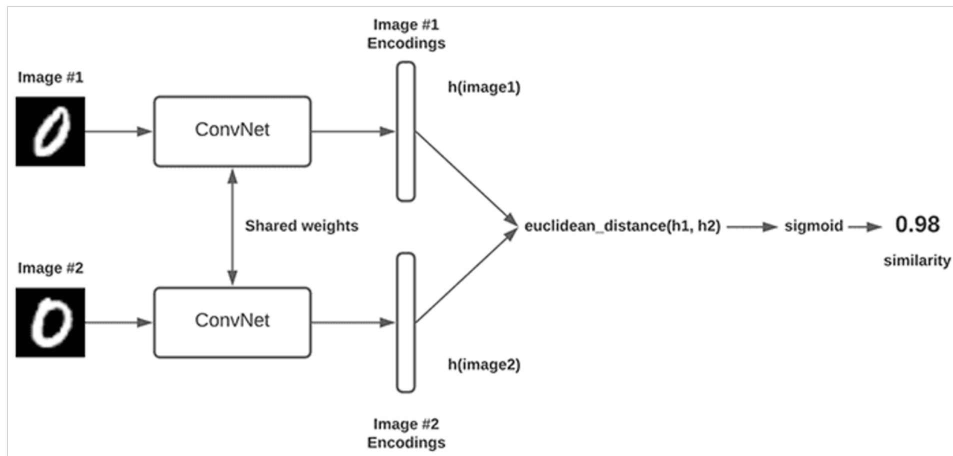
11.3.1. Sziámi hálók

Az egyik legegyszerűbb kontrasztiív tanuló architektúra az úgynevezett sziámi hálózat, amely igazából egy közönséges konvolúciós neurális háló, amely egyszerre két bemenetet kap, melyekből jellemző vektorokat generál. Ezeket a jellemző vektorokat ezt követően összehasonlítjuk (például az euklideszi távolságukat vesszük), majd erre egy szigmoid nemlinearitást téve osztályozási feladatot alkotunk. Ekkor persze igazából a negatív példák (nagy euklideszi távolság) fognak 1 körüli értéket kapni, míg a pozitívak (0 közeli távolság) 0.5-öt, ez azonban így is egyszerűen tanítható.

Egy kifejezetten nagy problémája azonban ennek az architektúrának, hogy két véletlen kép összehasonlításából csak nagyon kevés információt kap a háló arra vonatkozóan, hogy minek kellene mire hasonlítani. Éppen ezért gyakori használni az úgynevezett triplet (hármasikrek) loss-t, melynek lényege, hogy a hálóba nem két, hanem három képet adunk: egy referenciát, egy pozitív, és egy negatív példát. Így biztosítjuk, hogy minden iterációban a háló mindkét féle tanításon átesik: azt is megmondjuk neki, hogy minek kellene hasonlítani, és, hogy minek nem.

Ezzel azonban még mindig akad egy probléma: A kontrasztív tanítás során általában jelentősen több a negatív példák, mint a pozitív példák száma, a triplet loss, viszont ezeket egyenlő súllyal kezelni (hiszen mindig 1-1 példát adunk mindkettőből). Ez úgynevezett adat imbalance problémához vezethet. Éppen ezért a gyakorlatban általában egy egész minibatchnyi képet adunk át a hálónak, és minden lehetséges képpár között számolunk egy losst, attól függően, hogy azok pozitívak vagy negatívak. Ez felfogható a triplet loss egyfajta általánosításának.

Erre egy jó példa az úgynevezett InfoNCE hibafüggvény, amely feltételezi, hogy a batchben minden képre pontosan egy pozitív példa létezik (ezt a minibatch összeállításakor tudjuk biztosítani), így



11.9. ábra. *A sziámi hálók elve.*

a kérdés visszavezethető egy egyszerű osztályozási problémára: Az adott képhez a batchből melyik másik elem a pozitív? Mivel itt egy egyszerű N osztályú osztályozási problémánk adódott, ezért ezt minden további probléma nélkül tanítható a már megismert kereszt-entrópia hibafüggvény segítségével. Érdemes megjegyezni, hogy az InfoNCE loss általánosítható arra az esetre is, ha több pozitív példa van egy minibatchben.

Fontos azonban megjegyezni, hogy még így is adódik egy meglehetősen nehéz probléma a tanulási folyamattal, mégpedig az, hogy a negatív minták nagyon nagy része könnyű, úgynevezett easy negative, mivel teljesen más tartalmú képeket szinte bármilyen neurális háló képes a jellemző tér távoli pontjaiba képezni. A tanulás igazi sikere azon múlik, hogy a hasonló, de más szemantikai tartalmú, úgynevezett hard negatív képeket is sikerül jól elkülöníteni. Ezek a példák azonban relatíve ritkák, így a véletlenszerűen mintavételezett batchekben feltehetően az easy negatívok fognak dominálni. Erre egy megoldás az úgynevezett hard negative mining, melynek lényege, hogy a tanítás közben szándékosan keresünk nehéz negatív példákat (pl. amiket a háló korábban elrontott), és ezeket lényegesen nagyobb eséllyel adjuk be újra tanításra.

11.3.2. SimCLR

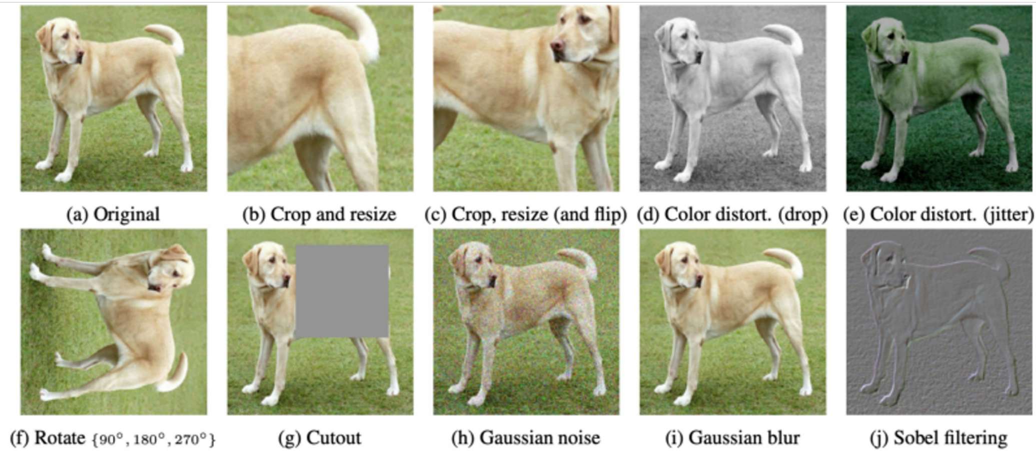
Az eddig tárgyaltak során taktikusan elhallgattam azt a kérdést, hogy hogyan döntjük el, hogy két példa pozitív vagy negatív kapcsolatban áll. Ezt azért tettem, mert a legtöbb korai kontrasztív megoldás ezeket a képek osztálya alapján döntötte el, vagyis egy felcímkézett, felügyelt tanulásra is használható adatbázist használtak. Az eredményeik így is figyelemre méltóak, hiszen megmutatták, hogy az osztályzás megtanulható a címkék explicit előírása nélkül is, azonban ez még igazi önfelügyelt tanulásnak nem nevezhető.

A következő módszer, a Simple Framework for Contrastive Learning of Representations (SimCLR), már nem így működik; Ez a módszer az adataugmentáció technikáját használja fel a pozitív minták generálására, vagyis egyes képek akkor is negatívak lesznek, ha ugyanaz az osztály található rajtuk, de nem ugyanabból a képből lettek létrehozva. Ennél a módszernél is úgy állították össze a minibatch-et, hogy minden képnek pontosan egy pozitív párja legyen. A SimCLR-nek egy fontos újítása volt még, mégpedig egy újszerű hasonlósági függvény, az úgynevezett cosine similarity használata. A cosine similarity egyszerűen nem más, mint két vektor által bezárt szög koszinusza, vagyis:

$$CosSim(x, y) = \frac{x^T y}{||x|| \ \ ||y||} \tag{11.4}$$

A cosine similarity egyik jelentős előnye az euklideszi (vagy más hasonló) távolság mércével ellentétben, hogy amennyiben a költségfüggvényben valamilyen okból elkezd az egyik tag (vagy a

pozitív példák közti távolság minimalizálása, vagy a negatívok maximalizálása) dominálni, akkor a háló elkezdheti csak ezt a tagot optimalizálni a másik kárára. Ennek eredményeképp vagy nullvektort kapunk minden adathoz, vagy kvázi végtelen nagyot (és akkor ezek távolsága is vagy nulla, vagy végtelen). A távolságmércékkel szemben a cosine similarity a vektorok bezárt szögét nézi, így nincs ilyen degenerált megoldása.



11.10. ábra. *A SimCLR során használt augmentációk.*

Érdemes még megjegyezni, hogy a SimCLR megoldás akkor működik igazán jól, hogy ha sok, és meglehetősen erős augmentációt használunk, és így a pozitív példák meglehetősen különböznek. Ellenkező esetben a feladat könnyen triviálissá válhat. Szintén fontos a jó tanuláshoz, hogy meglehetősen nagy minibatch méretet használjunk, hiszen így sokkal nagyobb eséllyel lesznek érdekes negatív minták a batchen belül. Természetesen a hard negative mining itt is használható (és célszerű is).

11.3.3. MoCo

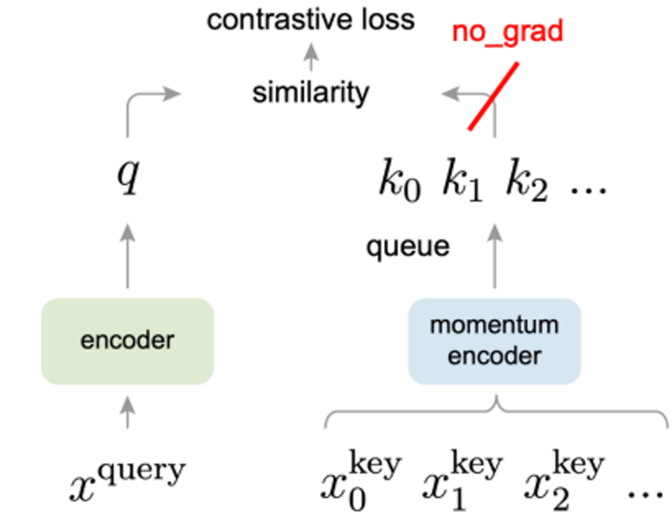
Egy érdekes megoldás a Momentum Contrastive Learning, vagy MoCo, mely egy ügyes trükkel orvosolja a tanulás egyik problémáját. A működés során vagy egy query példánk, és az ebből kinyert jellemző vektorokat hasonlítjuk a többi bemenetből (ezeket key-nek hívjuk) kinyertekkel. A korábbi módszerekkel szemben azonban ezt nem ugyanaz a hálózat teszi meg, hanem a query és a key objektumoknak különböző hálózata van. A tanítás során mi csupán a query encoder hálózatát tanítjuk a gradiens módszerrel, a key encodere pedig egy momentum update szabály alapján változik az alábbi módon:

$$\vartheta_k = m\vartheta_k + (1 - m)\vartheta_q, \tag{11.5}$$

vagyis a key encoder paramétereinek új értéke az előző érték, és a query encoder paramétereinek súlyozott átlaga, ahol a relatív súlyt a momentum tag m határozza meg. Ennek az értelme az, hogy a key-ek felé nem végzünk backpropagation-t, vagyis azokat az aktivációkat nem kell benn tartani a GPU memóriában. Ezzel lehetővé tesszünk, hogy nagyon sok negatív példát - és így sok érdekeset - adjunk a hálózatnak anélkül, hogy GPU memória problémákba ütköznénk.

11.3.4. DINO

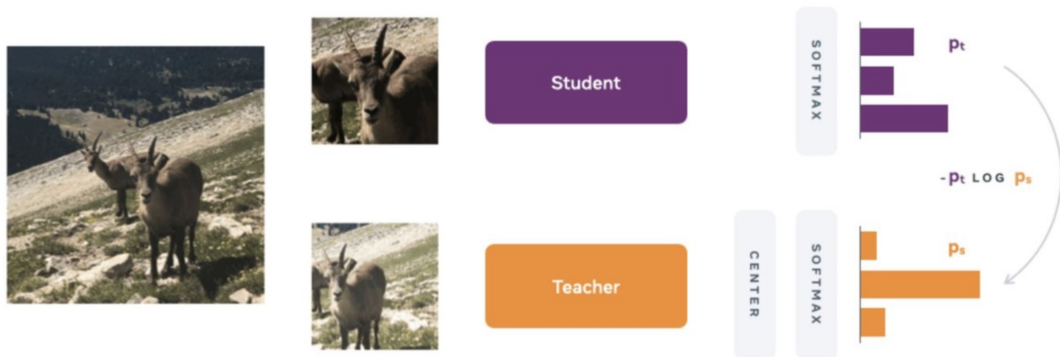
Érdemes még megemlíteni a DINO megoldást, amely Vision Transformerek önfelügyelt tanulására alkalmazható módszer. A DINO működésének lényege, hogy két hálózatunk van, egy student és egy teacher hálózat, mindkettő egy osztályozási feladat elvégzésére alkalmas. A bemenetként használt címkézetlen képekből először lokális (< teljes kép 50%-a), illetve globális (> teljes kép 50%-a)



11.11. ábra. *A MoCo módszer elve.*

kivágásokat (crop-okat) készítünk, majd ezeket erősen augmentáljuk. Ezt követően a globális crop-okat a teacher, míg a lokálisakat a student hálózatnak adjuk.

A student hálózatot arra tanítjuk, hogy kimenetén adja ugyanazt a valószínűségi eloszlást, mint a teacher, míg a teacher-t az előző módszerben használt momentum módszerrel tanítjuk. Ennek következtében a hálózatok megtanulnak egy olyan belső reprezentációt készíteni, ami jól általánosít a lokális és a globális nézetek között, így rendkívül jól használható más feladatok elvégzésére is.



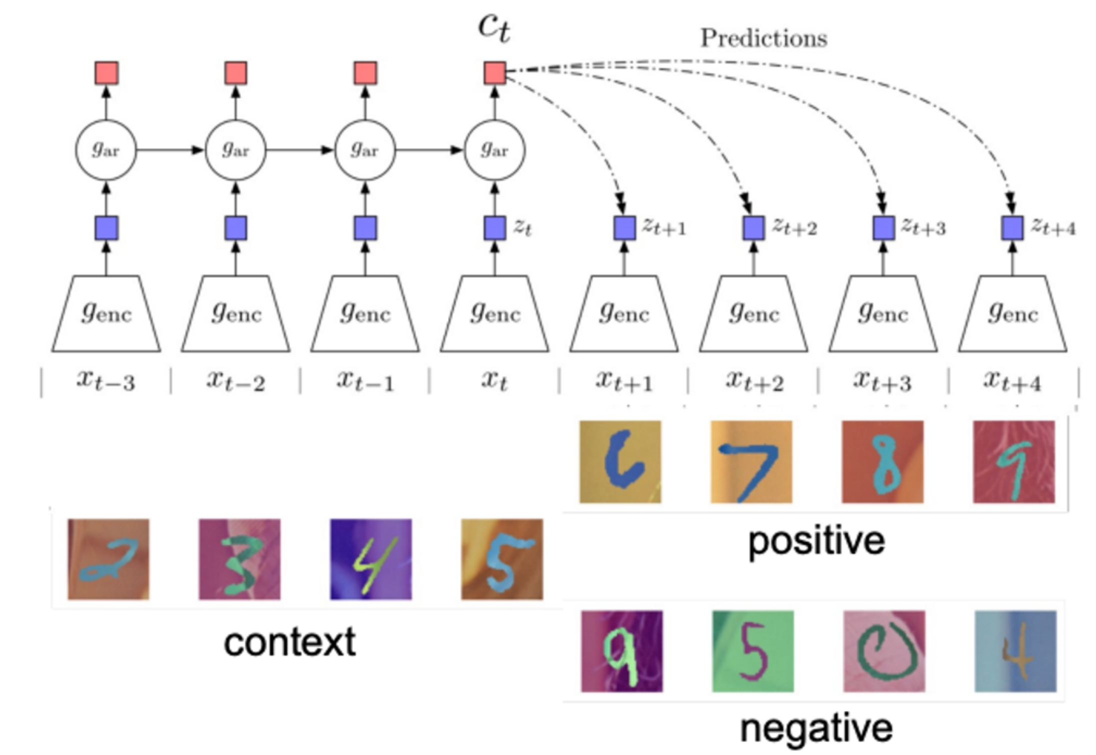
11.12. ábra. *A DINO architektúra.*

A módszer viszont kifejezetten érzékeny lehet a generatív modelleknél is jelentkező mode collapse jelenségére, azaz semmi nem akadályozza meg a módszert abban, hogy minden bemenetre ugyanazt a jellemző vektort becsülje. Ezt a problémát két heurisztika együttes alkalmazásával kerüljük el; Ezek közül az egyik a centering, a másik pedig a sharpening névre hallgat.

A centering - nevéhez híven - a hálózatok által predikált valószínűségekből levonja azok mozgóátlagát (ezt általában exponenciális átlagolással számítjuk), így ha a hálózat mindenre ugyanazt predikálná, akkor az folyamatosan levonásra kerülne, így a hálóból mindig közel nulla jóság értékek jönnének ki. A sharpening ezt követően a SoftMax bemenetét egy relatíve nagy pozitív számmal súlyozza, amivel a SoftMax után kapott eloszlás sokkal "hegyesebb" lesz, hiszen a nagy szorzó a bemeneti jóságok közti apró különbségeket felnagyítja. Mivel a hálózat nem képes minden bemenetre *pontosan* ugyanazt becsülni, ezért a kimenetek közti apró különbségek felnagyítva eltérést fognak okozni a kimeneten.

11.3.5. CPC

Érdemes még végezetül megjegyezni egy másik érdekes módszert, mégpedig a Contrastive Predictive Coding (CPC) névre hallgató eljárást. Ez a módszer nem képeken, hanem képek szekvenciáin végez kontrasztív tanulást, mégpedig oly módon, hogy egy szekvencia első fele alapján elvárjuk tőle, hogy meg tudja különböztetni a szekvencia helyes második felét egy helytelen folytatástól. Ennek a módszernek a lényege, hogy ne csak képi, hanem időbeli jellemzőket is képesek legyünk tanulni, melyek videókra is működő kontrasztív tanuláshoz elengedhetetlenek.



11.13. ábra. *A CPC módszer elve.*

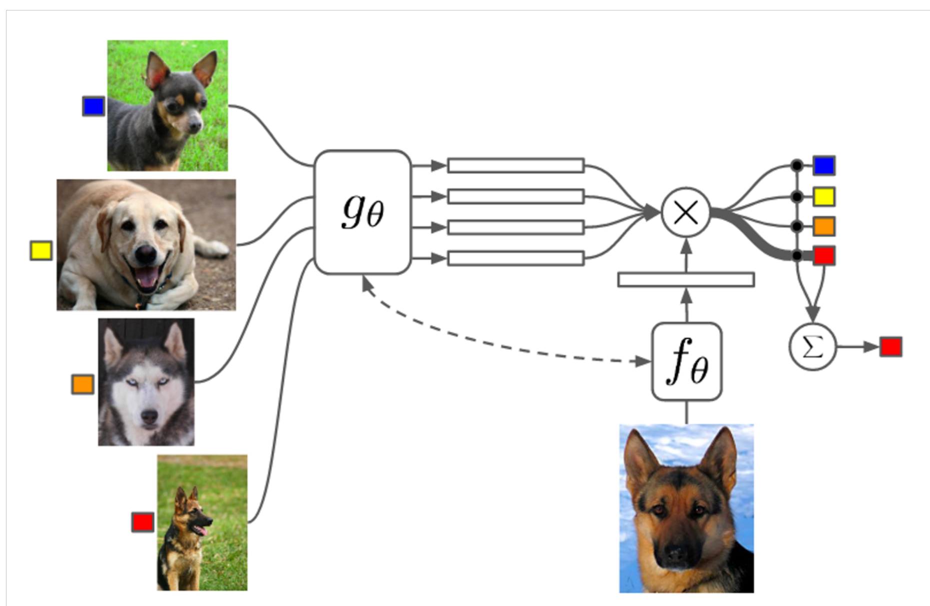
11.4. Few-shot tanulás

Végezetül érdemes megjegyezni, hogy ugyan a generatív módszerek egy fontos előnye a kontrasztívokkal szemben, hogy a hard negative példák nem okoznak nehézséget, a kontrasztív tanulásnak több előnye is van. Ezek közül az egyiket már a bevezető részben említettük: Ezek a hálózatok explicit módon vannak hasznos reprezentációk tanulására késztetve, míg a többi esetben csak feltételeztük, hogy a megtanult jellemzők tényleg jól általánosíthatók.

Egy másik előnye, hogy a kontrasztív módon megtanult jellemzők könnyedén felhasználhatók arra, hogy az ember egy olyan képességét megvalósíthassuk neurális hálók segítségével, amire a hagyományos felügyelt tanuló algoritmusok egyszerűen képtelenek. Ez a képesség mégpedig az úgynevezett few-shot tanulás, vagyis az a képesség, hogy egy új - még korábban nem látott - osztályt csupán néhány felcímkézett példa bemutatása után megtanuljunk felismerni.

A few-shot learningnek több alesete is ismert, a legnehezebb, ún. zero-shot learning esetében a hálónak egyetlen példát sem mutatunk be az új osztályból, hanem mindenféle tanítás nélkül kell adaptálódnia az új osztályhoz, míg a one-shot learning esetében pontosan egy felcímkézett képet kap. Általánosabban a few-shot leaning algoritmusok kevés (few), de egynél több címkézett kép alapján tanulják meg az új osztályt.

A few-shot learning egyik legegyszerűbb módszere az úgynevezett Matching Networks, melynek működése során felhasználunk egy kontrasztív módon betanított enkódert, mely a bemenetre kapott osztályozandó query kép jellemzőit előállítja. Adott még egy key adathalmaz, melyben a tanításhoz használt néhány kép van, ezek jellemzőit is kinyerjük az enkóder segítségével. Ezt követően a query kép jellemzőit összehasonlítjuk a key set jellemzőivel, és a hasonlóságok alapján végzünk osztályzást (pl. többségi szavazással, vagy súlyozott szavazással).



11.14. ábra. *A Matching Network módszer elve.*

IV. rész

Alkalmazások

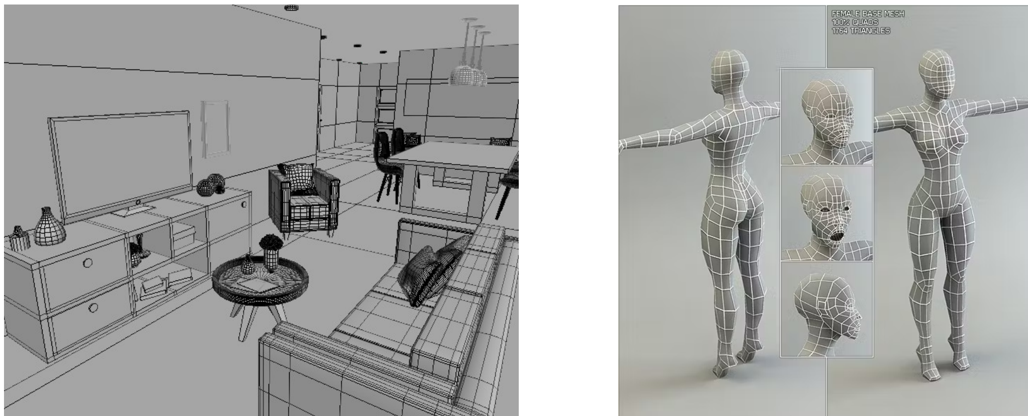
12. fejezet

Neurális renderelés

Napjainkban a 3D virtuális színterek, virtuális világok egyre nagyobb szerepet kapnak az életünkben. Ilyennel találkozhatunk a virtuális valóság (VR) alkalmazásokban, modern számítógépes játékokban, CGI-t használó filmekben, vagy akár egy való életbeli jelenet, látkép digitális rekonstrukciója során. A virtuális színterek renderelése hagyományos számítógépes grafikai eszközökkel azonban nem mindig hatékony, hiszen minden apró részletet (textúrát, megvilágítást, fizikai jelenségeket, stb.) nekünk kell megadni. Ezen segít a neurális renderelés, amelynek során a klasszikus számítógépes grafikai eszköztárat neurális hálózatokkal egészítjük ki [57, 58]. A következőkben arról lesz szó, hogy milyen lehetőségek vannak a mélytanulás alkalmazására a renderelés során, illetve milyen alkalmazási területeken számít sokat ez az új technológia.

12.1. Hagyományos renderelés

A 3D virtuális világok létrehozása egy komplex folyamat. A színtér minden objektumát egy sokszögháló reprezentálja, amely annál több sokszöget tartalmaz, minél részletesebben szeretnénk megjeleníteni az adott objektumot. Ezt a hálót vagy egy művész modellezi le, vagy egy valós objektum 3D szkenneléséből kapjuk meg. Mindkét megoldás elég időigényes lehet.

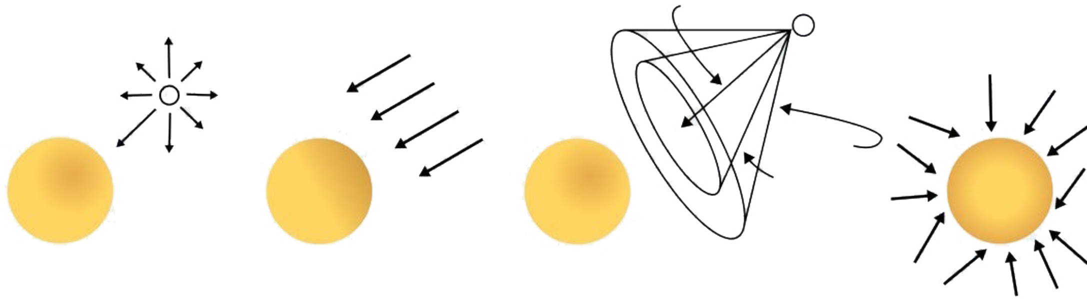


12.1. ábra. *3D virtuális világok felépítése sokszöghálókból.*

A sokszögháló csak az objektum geometriáját adja meg, de a megjelenítéshez hozzátartozik a felületén található anyag, textúra is. Az anyagtulajdonságok (például a szín, a simaság és a fényvisszaverési, fénytörési képesség) határozzák meg, hogy bizonyos fényviszonyok és kameraállás mellett hogyan néz ki az objektum. Ezeket a tulajdonságokat manuálisan kell megadni a színtér minden felületéhez. Manuálisan kell még megadni a fényforrásokat, megvilágítást is a világunkhoz.

A renderelés folyamata során a bemenetként szolgáló háromszöghálókból, anyagtulajdonságokból, fényforrásokból, illetve kameraállásból előállítunk egy képet a színtérről a fizika törvényei alapján.

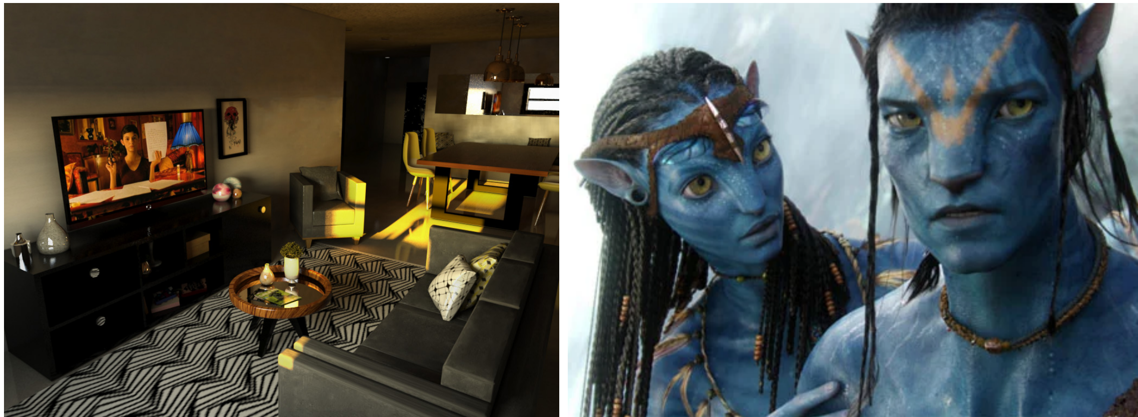
12.2. ábra. *Anyagtulajdonságokat meghatározó* material map *példák.*



12.3. ábra. *A fényforrások négy típusa: pont, irány, spot, ambiens.*

Ehhez a leggyakrabban használt algoritmus a *ray tracing*, amely a virtuális képernyő minden pixelén keresztül indít egy-egy sugarat és kiszámítja a rajta látható objektum színét. Ehhez, ha az eltalált objektum tükröző vagy fénytörő anyagból van, további sugarakat is indíthat.

Ez a renderelési folyamat lenyűgöző eredményeket tud elérni, viszont hatalmas mennyiségű munka szükséges minden objektum és minden anyag kifejezett meghatározásához, valamint a valósághű megjelenítéséhez óriási számítási kapacitás szükséges. Ez elvezet bennünket a kérdéshez: mi lenne, ha nem kellene minden tárgyat meghatároznunk és minden fényvisszaverődést kiszámítanunk? Erre nyújt megoldást a neurális renderelés.



12.4. ábra. *Példák renderelés eredményeire.*

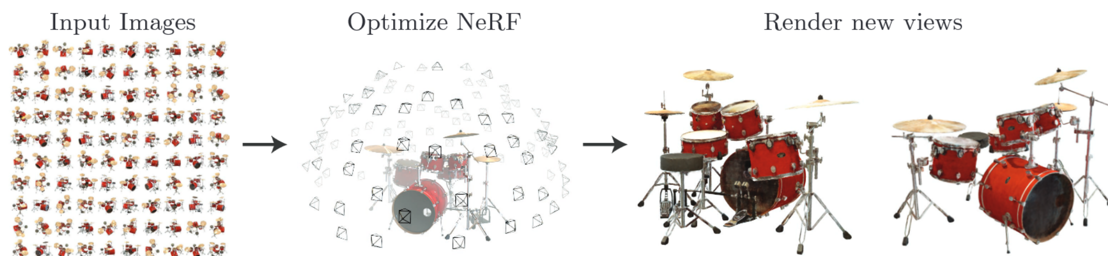
12.2. Neurális renderelés

A neurális renderelés során a fotorealisztikus képet egy neurális hálózat állítja elő. Ehhez a megközelítéshez nem kell kiszámolnunk a tárgyak és a fény közötti kölcsönhatásokat, mert azokat a tanító adatokból fogja elsajátítani a modellünk.

12.2.1. Új nézetek szintézise

A neurális renderelés egyik leghasznosabb felhasználása az új nézetek szintézise (novel view synthesis) [59]. Ennek lényege, hogy a háló megtanul egy 3D színteret tetszőleges nézőpontból renderelni.

A tanításhoz az adott színtérről különböző (de korlátos számú) nézőpontból állnak rendelkezésre képek, amelyeken adott (vagy valamilyen módon közelített) a kamera állása. A színteret egy térfogati adatként fogjuk fel, amit a kamera nézeti sugara pásztáz át. A pozíciót (3D adat) és a nézeti irányt (2D adat) összefűzve kapjuk meg a háló bemenetét. A háló ezután létrehozza a jelenet 3D-s ábrázolását, ahol a 3D-s tér minden pontjához hozzárendeli az ott található térfogati sűrűséget (átlátszóság) és a nézeti iránytól függő RGB színt.



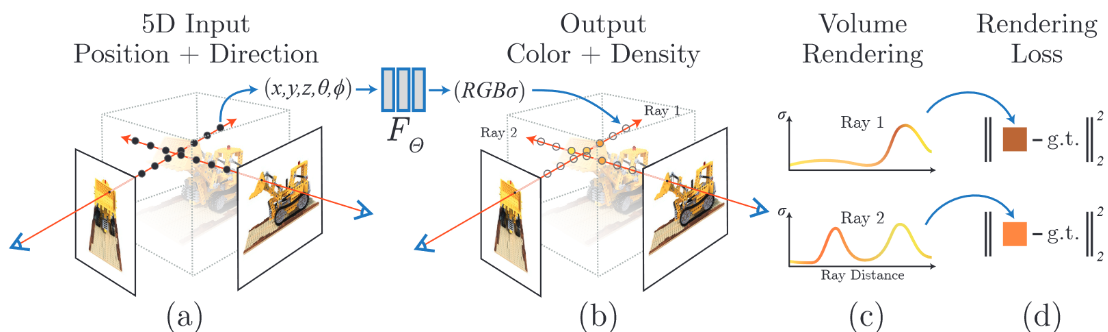
12.5. ábra. *Új nézetek szintézise neurális radiancia mezővel (NeRF-el).*

A neurális háló ebben a feladatban egy egyszerű teljesen összekötött háló (Multi-Layer Perceptron, MLP). Mivel a kimeneten tetszőleges valós nézeti szögben képes előállítani a színt és a térfogati sűrűséget, ezt az ábrázolást neurális radiancia mezőnek (neural radiance field, NeRF) nevezik.

Egy neurális radiancia mező renderelése a következő lépésekből áll:

1. kamera sugarakat küldünk a színtérre, hogy 3D mintapontokat kapjunk,
2. ezeket a pontokat a hozzájuk tartozó 2D nézeti iránnyal felhasználjuk bemenetként a neurális hálóhoz, ami kimenetként színeket és sűrűségeket állít elő,
3. hagyományos térfogati rendereléssel akkumuláljuk ezeket a színeket és sűrűségeket egy 2D képpé.

Ez a folyamat differenciálható, ezért használhatjuk hozzá a gradient descent algoritmust. Az optimalizáció során a megfigyelt képek és a neurális radiancia mező reprezentációból renderelt képek közötti eltérést, rekonstrukciós hibát minimalizáljuk. Ennek a hibának a több nézetben történő minimalizálása arra ösztönzi a hálót, hogy a színtérről koherens modellt állítson elő.

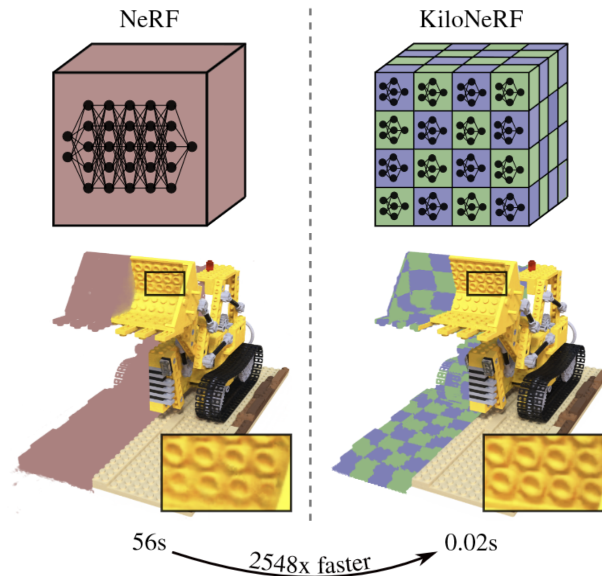


12.6. ábra. *Neurális radiancia mező tanítása.*

A neurális radiancia mezők már megtalálhatóak a Google Street View-ben is. Ez az új technológia lehetővé teszi a felhasználók számára, hogy úgy fedezzék fel a térképen látható helyszíneket, mintha egy videót készítő kamerát irányítanának. További részletek itt olvashatók.

12.2.2. Hatékonyabb reprezentáció

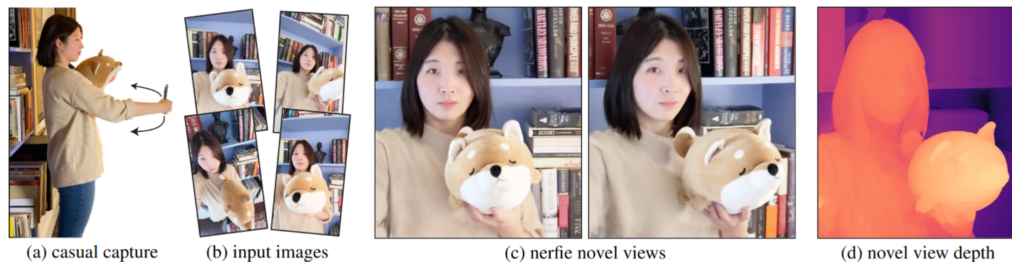
A neurális radiancia mezők renderelése alapvetően lassú, mert a benne található mély MLP hálót több milliószor is le kell kérdezni. Egy új megoldás, a KiloNeRF [60] ezen sebességbeli korlátokon javít: az egyetlen, óriási MLP háló helyett több ezer kisebb MLP-t használ. Minden MLP a teljes színtér egy-egy kisebb szegmensét képviseli, ezért sokkal kevesebb és gyorsabb lekérdezés fog vonatkozni rá. Ezt az oszd-meg-és-uralkodj stratégiát további optimalizálással kombinálva a renderelés három nagyságrenddel felgyorsul, miközben a képminőség változatlan marad.



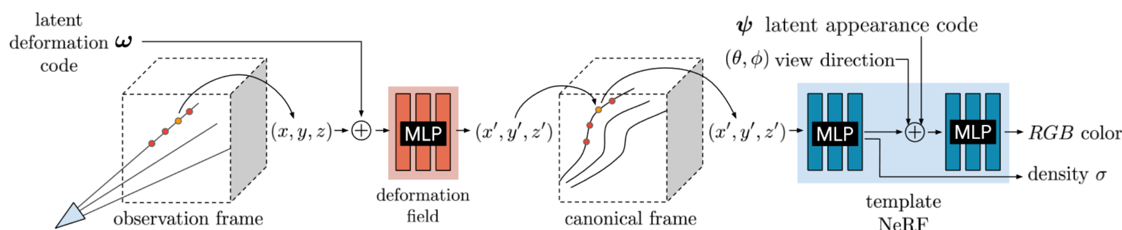
12.7. ábra. *A KiloNeRF alapötlete, hogy a NeRF-ben található óriási MLP hálót több ezer kisebb MLP-re osztja fel. Ezzel három nagyságrendbeli sebességjavulást lehet elérni.*

12.2.3. Deformálható színterek

A NeRF-ek egy lehetséges kiterjesztése deformálhatóvá teszi a renderelni kívánt színteret, vagyis az új nézőpont a tanító képek nézőpontjainak komplex nem-merev transzformációja által kapható meg (nagyítás/kicsinyítés, nyírás, dilatáció, stb.). Az első publikáció deformálható színterekről mobillal készített szelfikre épül és a megörökített személy arcát tudja rekonstruálni tetszőleges nézőpontból [63]. A létrehozott képeket „nerfie"-nek becézik mint a szelfi és a NeRF kombinációja. A módszer a NeRF-eket egészíti ki egy folytonos térfogati deformációs mezővel, amelyet a tanító képek alapján optimalizál. Ez a deformációs mező a térfogat minden pontját megfeleltet egy kanonikus 5D NeRF egy pontjának.

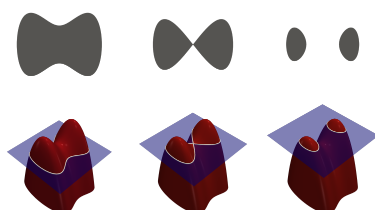


12.8. ábra. *A Nerfie modell különböző nézeti irányokból készült szelfik alapján képes új, deformált nézetekben reprodukálni a színteret.*



12.9. ábra. *A Nerfie modell működése. Minden képhez egy látens deformációs kódot (ω) és egy megjelenési kódot (Ψ) rendelünk. Nyomon követjük a kamerasugarakat a megfigyelt színtérben és a sugár menti mintapontokat egy kanonikus NeRF mezőjébe kódoljuk egy deformációs MLP háló és az ω kód segítségével. Minden transzformált mintára lekérdezzük a kanonikus NeRF értékét és integráljuk a kapottakat a sugár mentén.*

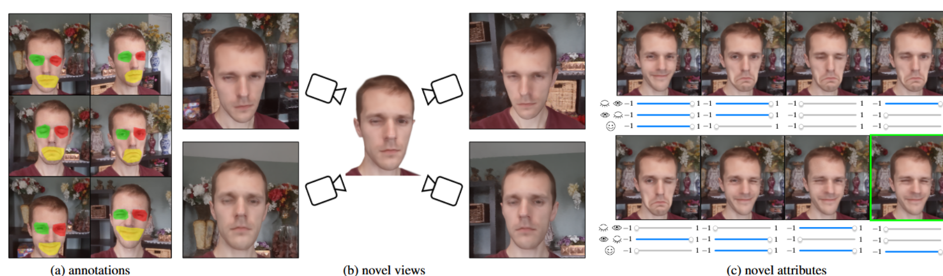
Ez a módszer azonban nem működik akkor, ha a megjelenített objektum topológiai változásokon esik át, mivel a topológiai változás egy diszkontinuiást eredményez a deformációs mezőben, viszont a deformációs mezőknek mindenhol folytonosnak kell lenniük. Egy újabb publikáció [64] úgy kezeli ezt a korlátozást, hogy a NeRF-eket egy magasabb dimenziós térbe emeli, és az egyes bemeneti képekhez tartozó 5D NeRF-eket ennek a hipertérnek egy-egy szeleteként értelmezi. Ehhez az inspirációt a level-set módszer adta, amely a felületek evolúcióját egy magasabb dimenziós felület szeleteiként modellezi. Az új, topológiát is modellező módszert HyperNeRF-nek hívják, és a korábbi, Nerfie módszerhez képest 4-8%-al kisebb átlagos hibát eredményez.



12.10. ábra. *A HyperNeRF alapja a level-set módszer, amely egy 2D alakzat evolúcióját (akár topológiai változással együtt) úgy írja le, hogy 3D térbe helyezi az alakzatot és annak síkmetszetei mutatják az eredeti 2D alakzat változását.*

12.2.4. Kontrollálható paraméterek

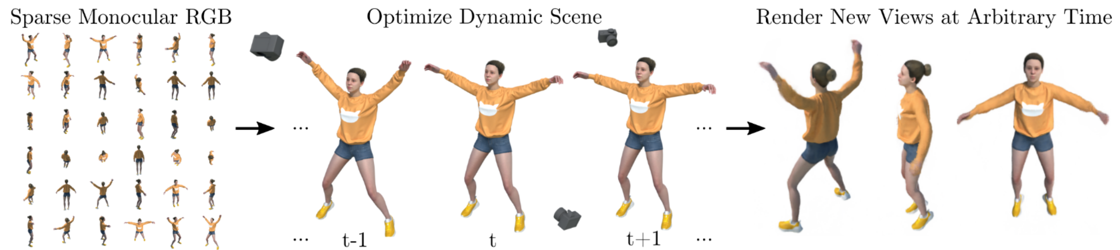
Neurális rendereléssel a színtér bizonyos paraméterei (például egy arcon a szemek, szájak mozgása) felhasználó által kontrollálhatóvá tehetők. Egy új módszer [65] lehetővé teszi a felhasználónak, hogy kijelölje, hogy a színtér mely részeit szeretné kontrollálni. Ehhez a tanító képeken kell néhány maszk annotációt elhelyeznie a kívánt helyeken. Az így kijelöl attribútumokat látens változóként kezeljük, amelyet a neurális háló visszafejt a színtér kódolása alapján. A tanítást követően pedig a háló képes lesz új, nem annotált képekből is kinyerni ezeket az attribútumokat és a felhasználói utasítások szerint változtatni őket a generált képen.



12.11. ábra. *Annotáció segítségével kijelölhetők azok a részletek, amiket kontrollálni szeretnénk.*

12.2.5. Dinamikus adatokra

Az eddig tanult neurális radiancia mezők alapvetően csak statikus színtérre alkalmazhatók, ahol az objektumok nem mozognak. Vannak azonban olyan esetek, amikor ez nekünk kevés, és az objektum (merev vagy nem merev) mozgását szeretnénk megörökíteni. Az ilyen esetekre alkották meg a dinamikus neurális radiancia mezőket (D-NeRF) [61].

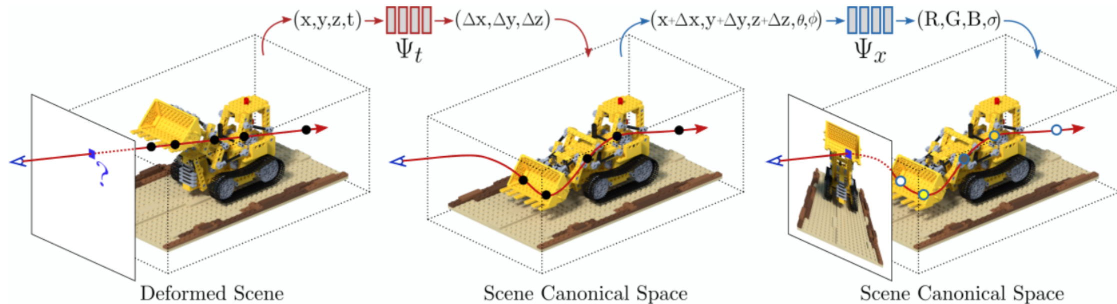


12.12. ábra. *A dinamikus neurális radiancia mező működése.*

A D-NeRF felépítése a statikus neurális radiancia mezőkéhez hasonló, de van egy további bemenete: az idő. Továbbá az architektúra egy helyett két MLP hálót alkalmaz:

1. egy hálót a deformált színtérnek egy közös kanonikus térbe való leképezésére („visszadeformálására" egy referencia állapothoz),
2. egy hálót a kanonikus térből az új nézeti kép szintéziséhez.

Ez a két háló egyidejűleg, egymástól függetlenül tanítható. A tanítás után a D-NeRF képes új képeket előállítani szabályozva a kamera nézetét és az időt, ezáltal az objektum mozgását is.



12.13. ábra. *A D-Nerf modellje két részre osztható: a Ψ_t deformációs hálóra, amely leképezi a színtér minden deformációját egy közös kanonikus térbe, illetve egy Ψ_x kanonikus hálóra, amely minden kamerából kiinduló sugárra visszafejti a térfogati sűrűséget és a nézetiirány-függő RGB színt, vagyis végrehajtja az új nézeti kép szintézisét.*

Ennek egy felhasználása a fotorealisztikus emberi avatar készítése, amit aztán különféle szerepekben (virtuális asszisztens, AR/VR játék karakter) fel lehet használni. Egy nemrég publikált cikk [62] egy olyan technológiát javasolt, ahol egy néhány perces videófelvétel után már fotorealisztikus avatart lehet készíteni a videón szereplő ember fejéről, ahol az avatar a mimikát, arckifejezéseket is pontosan tudja másolni. Ez az új modell a D-NeRF-ekre épít, amit kombinál egy parametrikus arcmodellel. Ez lehetővé teszi, hogy a korábban napokig, hetekig tartó tanítást kevesebb, mint 10 perc alatt végre lehessen hajtani.

12.3. Alkalmazások

A neurális renderelésnek számos alkalmazása van. Ezek közül eddig az új nézetek szintézisét és a fotorealisztikus emberi fej avatar készítését említettük elsősorban, de számos más példát lehet felhozni az új technológia használatára.

12.3.1. Újravilágítás

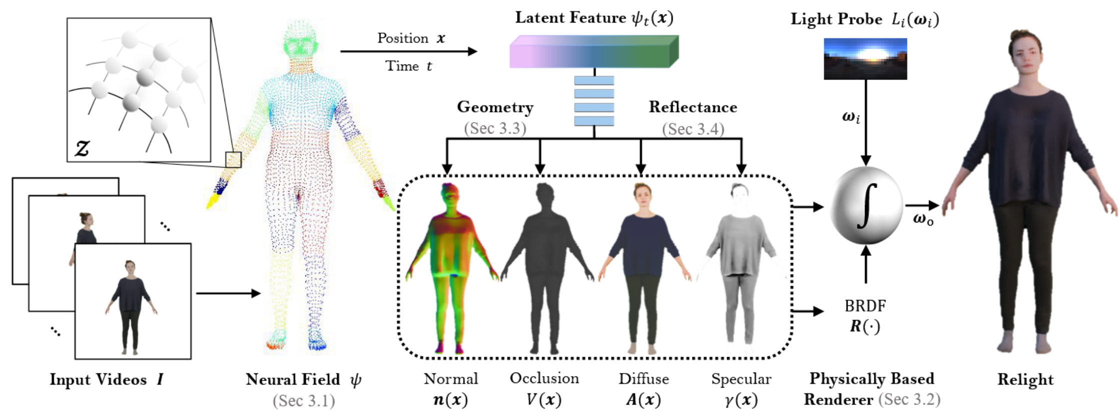
Az újravilágítás (angolul *relighting*) során a színtérről új fényviszonyok mellett renderelünk képet. Ez hasznos lehet számos számítógépes grafikai alkalmazásban, főképp vizuális effektusok létrehozására vagy kiterjesztett valóság megjelenítésére.

12.14. ábra. *Női arc újravilágítása, hogy illeszkedjen az új környezet fényviszonyaihoz.*

A képalapú újravilágítási technikák jellemzően különféle megvilágítású bemeneti képeket használnak arra, hogy megtanulják, hogyan nézhet ki a színtér új fényviszonyok mellett. Ennek hátránya a magas számítási költség, illetve a lassú adatgyűjtés. Az új, gyors újravilágítás algoritmusok mély neurális hálókat alkalmaznak tükröződési mezőkkel kombinálva, és általában korlátolt bemeneti adaton dolgoznak.

Egy példa az újravilágítás technikákra a Relighting4D keretrendszer [66], amelynek bemenete egy embert ábrázoló videó (egyszínű háttér előtt, más objektumok nélkül), ahol a megvilágítási körülmények ismeretlenek, és ebből hoz létre egy szabad nézőpontú, új megvilágítású videót. Alapötlete az, hogy az emberi test tér-idő függő geometriája és tükröződése felbontható normál, okklúzió (kitakarás), diffúz, illetve spekuláris térképeket reprezentáló neurális mezőkre. Ezek a neurális mezők tovább integrálhatóak egy fizikai alapú renderelési folyamatba, ahol a neurális mező minden csúcspontja (vertexe) elnyelheti és visszaverheti a környezet fényét. A teljes keretrendszer felügyelet nélkül tanul a videókból és a regularizációt fizikai alapú priorok biztosítják.

12.15. ábra. *A Relighting4D bemenetként egy videót vár, amin egy ember mozog semleges háttér előtt. A testből kinyeri a geometriát és a tükröződést, amely ezután felhasználható új megvilágítás melletti, szabad nézőpontú videók létrehozására.*



12.16. ábra. *Relighting4D: Vegyük a bemeneti videó frame-jét a t időpontban. A Relighting4D az embert egy Ψ neurális mezőként ábrázolja a látens Z vektorokon, amelyek egy deformálható emberi modellhez vannak rögzítve. A $\Psi_t(\mathbf{x})$ neurális mező értékét a 3D tér bármely* **x** *pontjában és t időpontjában látens jellemzőnek vesszük, amit egy MLP hálóba küldünk. Az MLP meghatározza a test geometriáját és tükröződését: a normál, az okklúziós, a diffúz, illetve a spekuláris térképet. Végezetül egy fizikai alapú renderelővel előállítjuk az új képet a kívánt megvilágítás mellett.*

12.3.2. Arc újraalkotás

Az arc újraalkotás (angolul: face reenactment) egy olyan arcszintézis feladat, amelynek célja egy kiinduló arcnak a formáját (póz, arckifejezés) átültetni egy célarca úgy, hogy a célarc identitása ne változzon. Ezt a feladatot neurális rendereléssel hatékonyan meg lehet valósítani, hiszen vannak algoritmusok a fejpozíció és az arckifejezés kontrollálására [67].

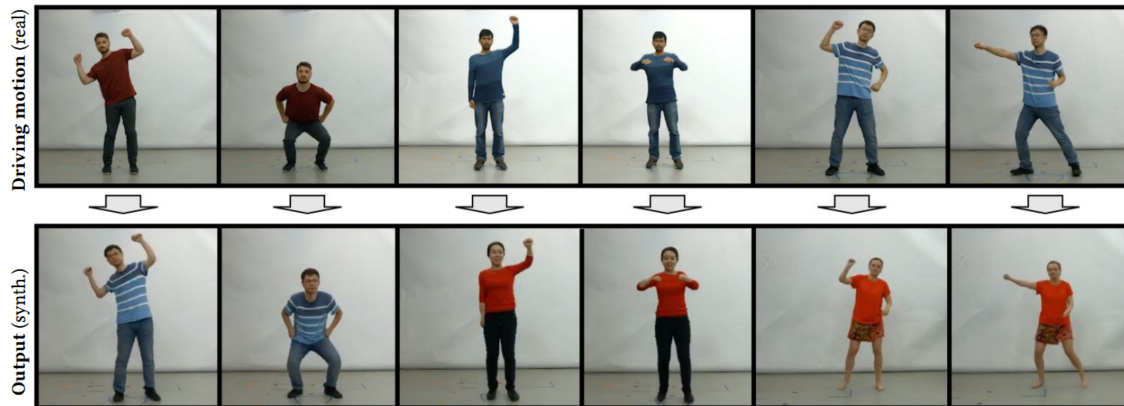
12.17. ábra. *Az arc újraalkotás célja egy kiinduló arcnak a formáját (póz, arckifejezés) átültetni egy célarca úgy, hogy a célarc identitása ne változzon.*

12.3.3. Test újraalkotás

A neurális renderelés lehetővé teszi a pózvezérelt videó- és képgenerálást, ahol a megjelenített ember által felvett póz kontrollálható. A test újraalkotás (body reenactment) ennek egy változata: lényege, hogy egy emberről készült kiinduló videón lévő mozgást átültetünk egy másik emberről készült célvideóra, a célszemély identitását megőrizve [68].

A test újraalkotás egy rendkívül bonyolult feladat, mivel az emberi test számos módon végezhet nemlineáris mozgásokat, nehéz garantálni a kimenet valósághű és fotorealisztikus megjelenését. Mindemellett a jelenlegi módszerek nagyrésze egyénspecifikus, vagyis csak egy adott személypárra működik és bármilyen új személlyel való munkához nagy mennyiségű tanítóadatra van szükség. Gyakoriak még a kimeneten a különféle artifaktok és sokszor a modellezés nem tudja jól kezelni az

egyes testrészek méretének skálázását. Összességében azt lehet mondani, hogy a test újraalkotás egy fontos feladat, de még szükség van kutatásokra a robusztus megoldás megtalálásához.



12.18. ábra. *A test újraalkotás célja egy kiinduló emberi test mozgását átültetni egy célszemélyre úgy, hogy a célszemély identitása ne változzon.*

További Olvasnivaló

[57] Ayush Tewari és tsai. „State of the Art on Neural Rendering". (2020). arXiv: 2004.03805.

[58] Ayush Tewari és tsai. „Advances in Neural Rendering". (2021). arXiv: 2111.05849.

[59] Ben Mildenhall és tsai. „NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis". (2020). arXiv: 2003.08934.

[60] Christian Reiser és tsai. „KiloNeRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs". (2021). arXiv: 2103.13744.

[61] Albert Pumarola és tsai. „D-NeRF: Neural Radiance Fields for Dynamic Scenes". (2020). arXiv: 2011.13961.

[62] Wojciech Zielonka, Timo Bolkart és Justus Thies. „INSTA: Instant Volumetric Head Avatars". *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2023. jún.

[63] Keunhong Park és tsai. „Deformable Neural Radiance Fields". (2020). arXiv: 2011.12948.

[64] Keunhong Park és tsai. „HyperNeRF: A Higher-Dimensional Representation for Topologically Varying Neural Radiance Fields". (2021). arXiv: 2106.13228.

[65] Kacper Kania és tsai. „CoNeRF: Controllable Neural Radiance Fields". (2021). arXiv: 2112.01983.

[66] Zhaoxi Chen és Ziwei Liu. „Relighting4D: Neural Relightable Human from Videos". (2022). arXiv: 2207.07104.

[67] Gee-Sern Hsu, Chun-Hung Tsai és Hung-Yi Wu. „Dual-Generator Face Reenactment". *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, 632–640. old. DOI: 10.1109/CVPR52688.2022.00072.

[68] Caroline Chan és tsai. „Everybody Dance Now". *CoRR* (2018). arXiv: 1808.07371.

13. fejezet

Orvosi alkalmazások

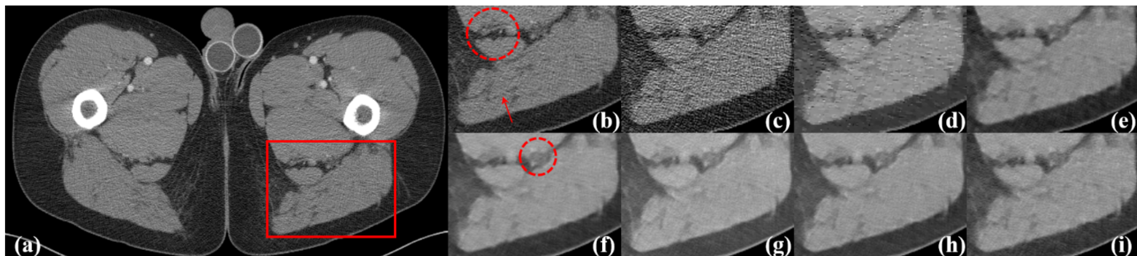
A neurális hálózatok technológiai fejlődése teret engedett az orvosi alkalmazások megjelenésének. Ezek az új algoritmusok némely területen már képesek a valós szakembereket is felülmúlni, azonban a nehéz validáció miatt egyelőre korlátozott a felhasználásuk. A következőkben azt fogjuk megnézni, hogy milyen fő alkalmazási területek vannak és milyen kihívásokkal jár a mesterséges intelligencia orvosi felhasználás esetén.

Jelen tárgy keretei között elsősorban az orvosi képalkotásra és vizuális megjelenítésre fókuszálunk, a mesterséges intelligencia más felhasználási lehetőségeitől (beteginformációk szemantikai feldolgozása, populációalapú predikció, stb.) eltekintünk.

13.1. Zajcsökkentés

Az orvosi képalkotás eredendően magas zajszinttel (rossz jel-zaj viszonnyal) küzd az emberi test komplexitása és árnyékolási képessége, illetve nehéz hozzáférhetősége miatt. A zajcsökkentés ráadásul nagy körültekintést igényel, hiszen a zaj mellett klinikailag jelentős struktúrákat is eltüntethet a képről, például parányi, kialakulóban lévő daganatokat. A hagyományos képfeldolgozó algoritmusok az ilyen komplexitású feladatoknál nem mindig elegendőek; a precizitást és megbízhatóságot egy megfelelően tanított mély neurális hálózat jelentősen tudja növelni [**Prabhat-2021**].

Példa: CT felvételek készítésekor a pácienst érő röntgensugár-mennyiséget igyekezni szoktak minél alacsonyabban tartani a káros élettani hatások elkerülése érdekében. Az alacsony jelszint miatt azonban a zajszint aránylag magas, vagyis az elkészült kép zajos lesz. Alacsony dózisú CT képek zajosságának csökkentésére számos módszer létezik, amelyek közül a legjobb eredményt a konvolúciós neurális hálózatok adják. Egy egyszerű, 3 rétegből álló CNN is már képes lehet jó eredményeket elérni [**Prabhat-2021**]. Természetesen rengeteg komplexebb megoldás is létezik, amelyek precízebbek és robusztusabbak a hagyományos CNN-eknél. Ilyenek lehetnek a különféle autoencoderek [**Liu-2018**] vagy a Wasserstein GAN [**Yang-2018**] is.



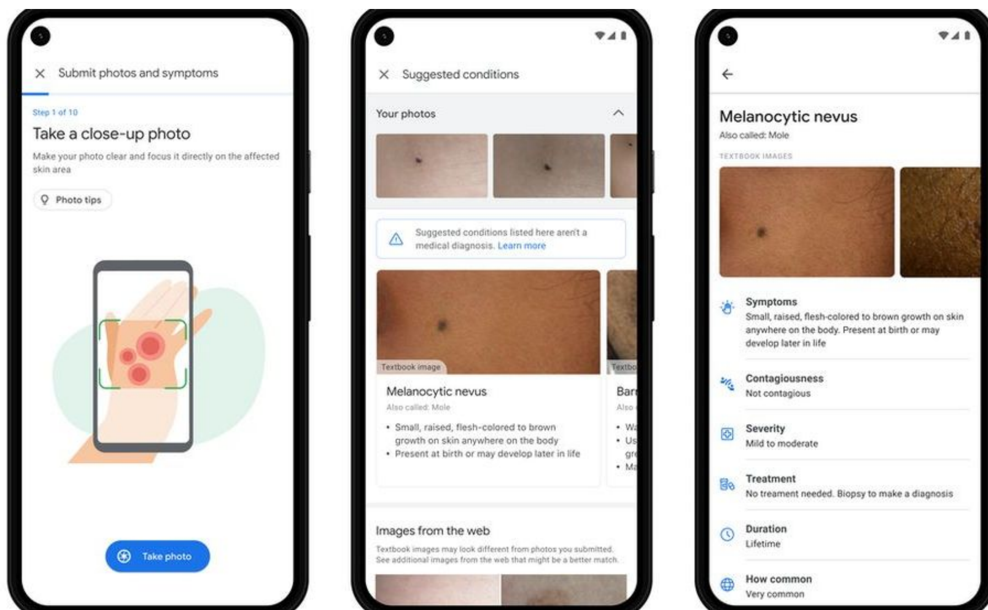
13.1. ábra. *Zajos CT felvétel, amely egy elváltozást mutat a beteg májában. Az alképekben különböző CNN architektúrák zajszűrési teljesítménye látható [***Prabhat-2021***].*

13.2. Detekció és osztályozás

Orvosi képalkotásban gyakran fordul elő, hogy egy rendellenességet, például daganatot keresünk a betegről készült képen. Ez a rendellenesség nem biztos, hogy könnyen észrevehető, főleg, ha a kiértékelő orvosnak több ezer képet kell átnéznie. Egy tipikus példa erre a tüdőszűrés, amely esetén rengeteg mellkas röntgen készül, viszont ezeket aránylag kevés szakembernek kell kiértékelnie. Ilyenkor jelenthet nagy segítséget egy neurális hálózat alapú automatikus kiértékelő, amely másodpercek alatt átnéz több százezer röntgen felvételt és képes megmondani, hogy melyeken sejt tüdőproblémákat. Egy további lépésben pedig a detekción felül osztályozást is végezhet a háló, vagyis azt is képes lehet megmondani, hogy milyen fajta és milyen stádiumú daganatot lát.

Detektálási feladatokra az orvosi képeknél is alkalmazhatóak az általános képekhez bevált architektúrák: hagyományos konvolúciós neurális hálózatok, az AlexNet, a VGG, a ResNet, stb.

Példa: A melanóma a bőrrák legsúlyosabb fajtája. Gyakran fejlődik ki anyajegyekből vagy van a felszínen anyajegyekhez hasonló kinézete. Kevés ember megy el azonban rendszeres bőrgyógyászati vizsgálatokra, ahol a melanómákat ki tudnák mutatni. A Google 2021-ben kiadott egy DermAssist nevű bőrgyógyászati (dermatológiai) AI asszisztenst, ami egy fénykép alapján megmondja, hogy egy anyajegy vagy bőrprobléma egészséges-e vagy rákos. Természetesen ez a predikció nem számít orvosi diagnózisnak, tehát a kétes eseteket érdemes megnézetni szakemberrel is. További információk: https://health.google/consumers/dermassist/.

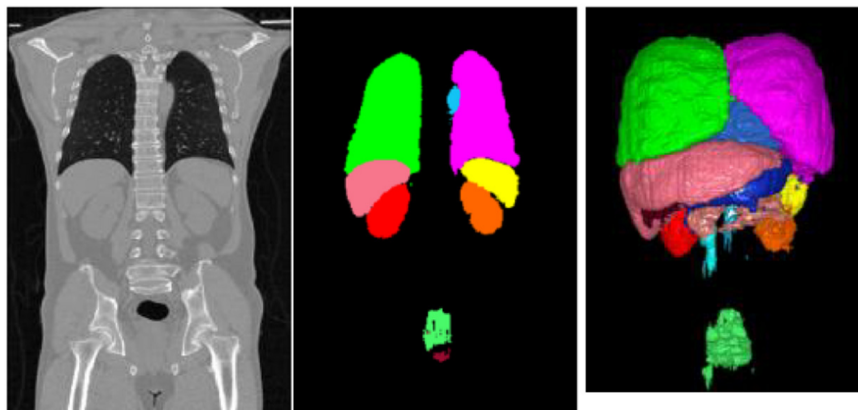


13.2. ábra. *Google DermAssist használat közben.*

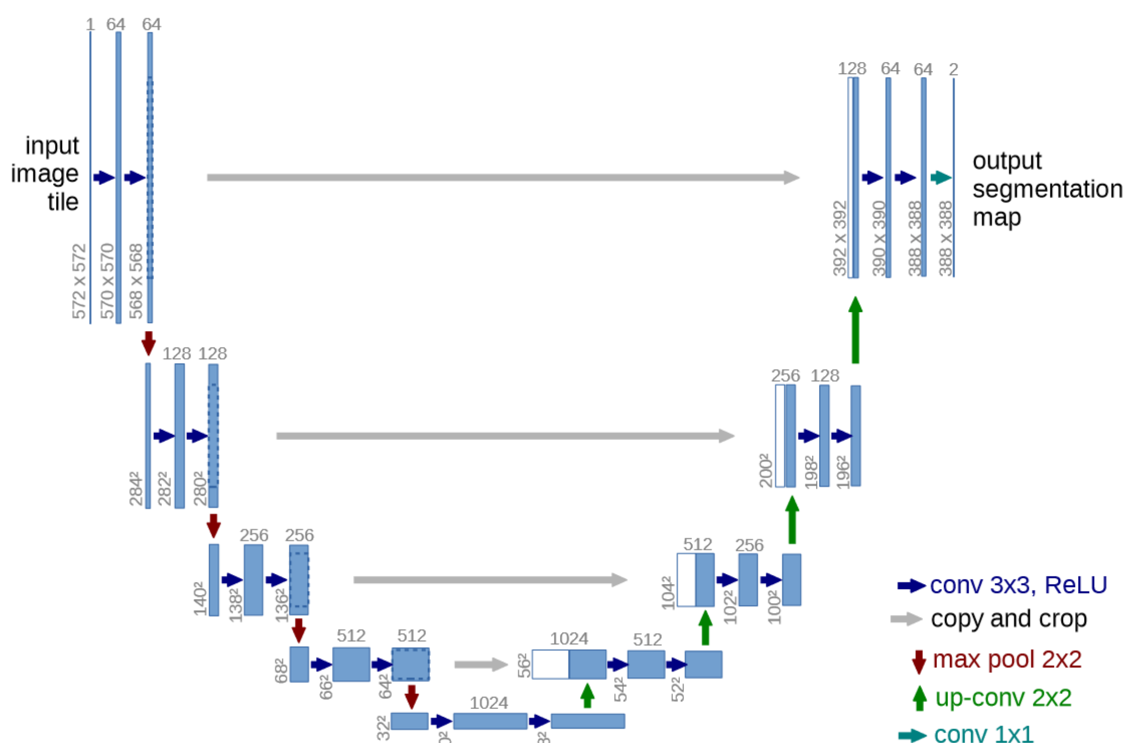
13.3. Szegmentáció

Az orvosi képalkotásban a szegmentáció célja a beteg egyes szerveinek, szöveteinek, illetve rendellenességeinek az elválasztása egymástól. Ez fontos lehet daganatok növekedésének ellenőrzéséhez, sérülések vagy rosszul gyógyult területek felfedezéséhez, de akár gyógyszerek adagolásának beállításához is. Az emberi test belső szegmentációja azonban egy rendkívül bonyolult feladat, mivel számtalan apró struktúra van benne és a szövetek átfedésben is vannak egymással a képen.

Orvosi képek szegmentálására 2015-ben alkották meg azt a neurális hálót, amelyet azóta is a leggyakrabban alkalmaznak: az *U-Net*-et [**UNet**]. Ez egy konvolúciós neurális hálózat, amely egy leskálázó és egy felskálázó részből áll. Az U-Net megnyerte a 2015-ös *Grand Challenge for Computer-Automated Detection of Caries in Bitewing Radiography* versenyt és a *Cell Tracking Challenge* verseny két legnehezebb kategóriáját, nagyban lekörözve a második helyezést.



13.3. ábra. *Szervek szegmentálása egy CT felvételen; 2D és 3D nézet.*

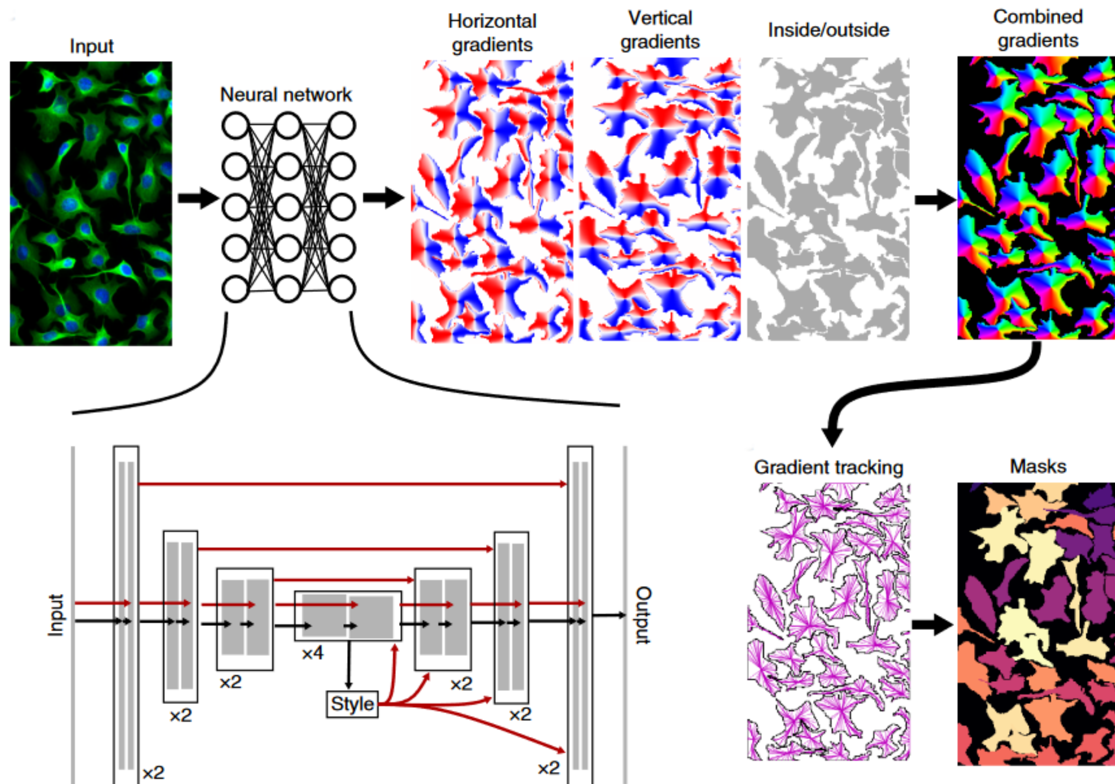


13.4. ábra. *Az U-Net architektúra, a leggyakoribb neurális háló orvosi alkalmazásokban.*

Példa: Számos biológiai alkalmazásnál szükség van a sejttestek, membránok és sejtmagok szegmentálására mikroszkópos képeken. Ez a feladatot oldja meg a Cellpose [**Cellpose**], ami egy általános, deep learning alapú keretrendszer sejtszegmentációra. A Cellpose belül az U-Net hálót használja.

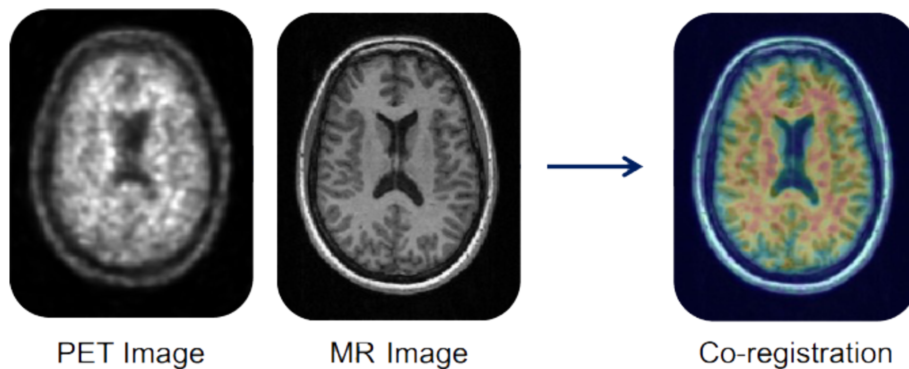
13.4. Képregisztráció és -fúzió

A különböző képalkotó modalitásokkal (pl. CT/MR/PET) készített felvételek különböző információkat tárnak fel a beteg testéről. Sok esetben hasznos ezeket a felvételeket egymással átfedésben látni, mivel így áttekinthető, hogy mi tartozik össze a két képen. Akár azonos modalitással, de eltérő időpontban készült képeket is hasznos lehet egymással átfedésbe rakni, például ha egy tumor változását szeretné az orvos szemlére venni. A felvételek összeillesztése azonban nem egyszerű feladat, mert például a páciensnek más a pozícionálása, orientációja, de időbeli eltérés esetén akár a testsúlyváltozás is zavarhatja a képregisztrációt.



13.5. ábra. *A Cellpose algoritmus sejtek szegmentációjára. A neurális háló egy U-Net.*

Példa: A pozitronemissziós tomográfia (PET) egy funkcionális képalkotó modalitás, amely a szövetek működéséről, anyagcseréjéről alkot képet. A testi struktúrákat azonban nem ábrázolja, ezért esetenként nehéz megmondani, hogy egy magasabb aktivitású régió pontosan melyik szervhez tartozik. Ha azonban a PET képet regisztráljuk egy CT vagy MR képpel, amely a szövetstruktúrát ábrázolja, akkor egyszerre láthatjuk a test szerkezetét és működését. Wei és társai [**Wei-2023**] egy kombinált módszert alkottak meg PET és CT képek fúziójára, amelyben a hagyományos radiológia és a deep learning módszereit ötvözték. Neurális hálózatnak a VGG 11-et használták. Lee és társai [**Lee-2022**] PET és MRI képek fúziójának előkészítéséhez használtak mélytanulást: mielőtt regisztrálták volna a két modalitás képeit, az MRI képet egy U-Net hálóval szegmentálták a feladat szempontjából érdekes rész kinyerése érdekében.



13.6. ábra. *PET és MR képek regisztrációja: a kimeneti képen látszanak az MR által meghatározott struktúrák, illetve a PET által meghatározott aktivitássűrűségek is.*

13.5. Kihívások

Mint láthattuk, a neurális hálózatok számos feladatra adnak jól teljesítő megoldást az orvosi képalkotás területén. Segítségükkel több betegség diagnosztizálható és az orvosok általános munkaterhelése csökkenthető. Azonban a neurális hálózatok technológiai fejlődése ellenére sem minden esetben egyértelmű és megbízható az eredmény, amit adnak.

A legnagyobb problémát a tanító adatok hiánya okozza. Mivel az orvosi adatok szigorúan bizalmasnak minősülnek, nem szokás őket közzétenni és terjeszteni, illetve publikus adatbázisokat létrehozni belőlük (ami adatbázisok vannak, azok jellemzően EU-s vagy akadémiai összefogások által jöttek létre). Az új tanító képek készítése is nehézkes, hiszen egy valós betegnek egy orvosi vizsgálaton vagy akár beavatkozáson kell átesnie hozzá. Ez az adathiány korlátozza a neurális hálózatok által elérhető pontosságot és megbízhatóságot.

Az érem másik oldalán viszont az etikai és jogi aspektusok állnak. Hosszú vitákat lehetne arról vívni, hogy etikus-e betegekről készült orvosi felvételeket akár névtelenül is kiadni egy neurális hálózat tanításához. Ráadásul előfordulhat, hogy a képpel együtt néhány alapvető személyes adatot is továbbítani kell; ha például szeretnénk biztosítani, hogy a háló egyenlő arányban lásson különböző nemű, etnikumú, korosztályú, stb. betegekről felvételeket, akkor ezeket az adatokat a képekkel együtt kell továbbítani. A képek tárolásának és terjesztésének szigorú jogi szabályozásra van szüksége, ez azonban még kialakulóban van.

Miután az adatok rendelkezésre állnak, ki kell választani a neurális hálózatnak az architektúráját. Minden feladathoz számtalan architektúra áll rendelkezésre, amelyeket ráadásul tetszés szerint lehet módosítani, vagy egy teljesen új architektúrát is ki lehet találni. Jelenleg nagyon kevés publikáció foglalkozik azzal, hogy milyen szempontok mentén lehet kiválasztani egy adott feladathoz legjobban illő neurális hálózatot.

Az architektúra kiválasztása utáni feladat a hiperparaméter-optimalizáció. A hiperparaméterek száma adott esetben elég nagy is lehet, és a paraméterek értelmezési tartománya, vagyis a keresési tér is számottevő. Sokszor nincs látható logika amögött, hogy egy adott feladatnál miért teljesít jól egy adott paraméterezés és rosszul egy másik; ezeket próbálgatás (trial-and-error) módszerrel lehet csak kideríteni. Mivel orvosi alkalmazásoknál akár életmentő lehet, ha a neurális hálózat a lehető legjobb paraméterezéssel dolgozik, ezért a hiperparaméter-optimalizáció megfelelő definiálása és végrehajtása egy újabb kihívást jelent.

Egy további fontos probléma a neurális hálók feketedoboz jellegű működése, vagyis hogy nem látunk bele a belső döntési mechanizmusaiba. Egy betanított neurális háló teljesíthet kiválóan, de nem ad magyarázatot arra, hogy mi alapján hozott meg egy adott predikciót. Emiatt nehéz megmondani azt is, hogy mennyire megbízható a jóslata. Ez sok visszásságot okoz orvosi alkalmazásoknál és rendkívül körülményessé teszi a klinikai validációt. Extenzív validáció nélkül pedig természetesen nem lesz engedélyezve az adott architektúra, illetve szoftver a gyakorlati felhasználásra.