Hatékony konvolúciós neurális hálózat tervezése osztályozási problémákra

Formanek András - mérnök informatikus BSc 4. évf
Konzulens: Hadházi Dániel

Méréstechnika és Információs Rendszerek Tanszék

Budapest, 2018. október 26.

Tartalom

1 Bevezetés

Az elmúlt néhány évben a konvolúciós neurális hálózatok megreformálták a képfeldolgozás területét. Sok probléma esetén utolérték és meghaladták a klasszikus, szakértői képfeldolgozó módszerek pontosságát és hatékonyságát. Egyelőre nem mutat semmilyen jel arra, hogy a neurális paradigma ilyen jellegű, egyre szélesebb körű alkalmazásának térnyerése ne folytatódna.

Azonban ezek az új megoldások gyakorlatilag feketedoboz jellegűek még azok számára is, akik értik, hogy általánosan hogyan működnek a neurális paradigma eszközei. Bár a tanulás folyamata világos, a megtanult tudás reprezentációja nehezen interpretálható. Így azon kívül, hogy meg tudjuk mérni, hogy az általunk gyűjtött, korlátos mintakészlet mekkora részére működik helyesen az adott megoldás, arra már nehezebb választ adni, hogy miért és milyen esetekben fog helytelen kimenetet előállítani. Ilyen jellegű bizonytalanságok nem engedhetőek meg biztonságkritikus rendszerek (például az önvezető autók) esetében, ahol emberi élet múlhat a rendszer hibás döntésein. Továbbá az is világossá vált, hogy nem csak a neurális paradigmán alapuló megoldások eseti hibáitól kell tartanunk, hanem lehetőség van a bemeneti képek olyan módon való manipulálására is, aminek hatására a megoldás helytelenül működik.

További probléma, hogy a state-of-the-art eredmények olyan neurális hálókon alapulnak, melyek sok, akár több száz réteggel és több millió konvolúciós szűrővel rendelkeznek. Ez a magas komplexitás a már említett interpretálhatóság problémája mellett felvet még egyéb hatékonysági problémákat is. Tipikusan százmilliós nagyságrendbeli paraméter tanulása és tárolása szükséges. Ennek következtében mind a tanítási, mind egy minta esetén a háló válaszának előállításhoz szükséges erőforrásigény is nagy. A nagyméretű és emiatt lomha hálók pedig ugyancsak hátrányt jelenthetnek bizonyos ipari felhasználásokban. Példaként itt is maradjunk az önvezető autóknál: ha a háló nem képes elég gyorsan döntést hozni, hogy kell-e fékezni adott szituációban, akkor alkalmassága megkérdőjeleződik. A hálók paraméterszámbeli és számítási erőforrásigény szerinti hatékonysága tehát ma kulcskérdés, és várhatóan az is marad az éles, gyakorlati alkalmazások esetén a jövőben is.

Munkám során[1] olyan módszereket vizsgáltam meg és tettem javaslatot módosításokra, hatásosságuk növelésére, melyekkel a fenti problémák enyhíthetők. Tételesen megvizsgáltam a tudás desztillálás alapú háló kompressziót, a Spatial Transformer réteg ilyen szempont szerinti alkalmazhatóságát, valamint javaslatot tettem a ma népszerű direkt regularizációs módszerek ilyen célból történő adaptálására. Minden taglalt módosítás hatását kvantitatívan és kvalitatívan is elemeztem, melyek eredményeképpen megállapítottam, hogy jelentős hatékonyságnövekedés érhető el a pontosság minimális szintű redukciója mellett.

[1] A dolgozatban taglalt kutatás az Európai Unió támogatásával, az Európai Regionális Fejlesztési Alap társfinanszírozásával valósul meg (EFOP-3.6.2-16-2017-00013).

A dolgozat felépítése az alábbi: a 2. fejezetben szakirodalomban publikált megközelítéseket tekintem át, a 3. fejezetben a tudásdesztilláció alapú paraméterredukciót mutatom be, a 4. fejezet a redundáns szűrések eliminálását taglalja, az 5. fejezetben összefoglalom az elért eredményeket és a hozzájuk kapcsolódó konklúziókat is levonom. A dolgozatnak részét képezi egy kétoldalas függelék is.

2 Szakirodalom áttekintése

2.1 Kapcsolódó kutatások

Ebben a fejezetben bemutatom a már ismert techológiákat, melyek célja konvolúciós neurális hálók (CNN), paraméterszámának csökkentése, vagy működésük átláthatóbbá tétele. Első esetben az erőforrás igény, második esetben az interpretálhatóság felől megközelítve értelmeztem, és próbáltam javítani a hatékonyságot.

2.1.1 Paraméter kvantálás és binarizálás

A paraméterek durvább granularitású ábrázolásával jelentősen redukálható a hálók tárolásához szükséges memória mérete. Ennek szélsőséges esete a binarizálás, amikor a háló összes súlya csak a 0 vagy az 1 értéket veheti fel, de ennél finomabb felosztás is elképzelhető. Például 8 bit, ahogy [7]-ben olvashatjuk. Itt az elsődleges cél az volt, hogy a modell CPU-n is futtatható legyen, és sikerült is 3-4-szeres gyorsulást elérniük, a lebegőpontos helyetti fixpontos ábrázolással, miközben a modell pontossága alig romlott. 16-bites ábrázolást is megvizsgált a szakirodalom: [8]. Összehasonlításképpen, az általában használt lebegőpontos float típus a C és C++ nyelvekben többnyire 32 bit, míg Pythonban 64 bit.

2.1.2 Paraméter nyesés és megosztás

A megközelítés lényege, hogy a hálónak valamilyen szempontból haszontalan részeit eldobjuk (nyesés) vagy más részeiből származtatjuk (megosztás), ezzel csökkentve a tanulandó paraméterek számát.

A javasolt metódusok között találunk olyat, ami az összes súlyra megvizsgálja, hogy annak eldobása mennyivel növelné a háló hibáját, majd az így sorba állított súlyok közül adott számút kinulláz [9]. Ez a módszer tehát a nyeséshez is használja az adathalmazt, hogy hibát tudjon számolni. Olyan módszer is létezik, ami súlyok helyett neuronokat céloz. A [10] javasolt módszere, hogy ha két neuron által tanult súlyok közel egyenlőek, akkor az egyik neuront dobjuk el, a másikat pedig, a következő rétegben vegyük figyelembe, az eddigi két neuron súlyának összegével. Látható, hogy ennek a módszernek a kimenete nem függ az adatoktól. Mindkét módszerrel jelentősen csökkenthető a paraméterszám.

2.1.3 Alacsony rangú faktorizáció és ritkítás

Az eddig említett módszerek többnyire csak a fully-connected rétegekre fókuszáltak. Ez érthető is, hiszen CNN-ek esetében is az utolsó pár réteg általában ilyen, és a tanulandó paraméterek nagy része ezekben lokalizálódik. Azonban a számítási idő nagy része még így is a relatíve lassan elvégezhető sok dimenziós konvolúciók számításából származik.

Például, ha minden 2D konvolúciót helyettesíteni tudunk két 1D konvolúcióval, szignifikáns gyorsulás érhető el, úgy, hogy közben a pontosságból nem veszítünk sokat. A [11] ezt az ötletet tárgyalja. Ha az intuitívabb (bemeneti csatorna szám) x (kimeneti csatorna szám) db 2D konvolúció elvégzése helyett 1db 4D konvolúcióként gondolunk egy konvolúcuiós rétegre, akkor ezt a 4D tenzort jó közelítéssel alacsonyabb dimenziójú mátrixok szorzatára

bonthatjuk és a konvolúciókat külön-külön végezhetjük el. Ezzel a módszerrel a [11] alapján 2-3-szoros gyorsulást érhető el a pontosság elhanyagolható mértékű csökkenése mellett.

2.1.4 Rétegengénti relevancia visszaterjesztés

A Layer-wise Relevance Propagation (LRP) [14] egy a neurális hálók tanítása során használt hiba visszaterjesztési (backward propagation) algoritmushoz hasonló technika, melyet kifejezetten magyarázási szándékkal terveztek.

A módszer a bemeneti kép pixeleihez súlyokat rendel, melyek a döntéshozás szempontjából az adott pixel relevanciáját minősítik. Működését tekintve az *l.* réteg relevanciáit ($R_i^{(l)}$) a következő réteg relevanciáinak ($R_j^{(l+1)}$) függvényében írjuk fel, majd a kimeneti rétegtől elindulva visszaterjesztjük a relevanciákat, amíg el nem érjük a bemenetet. A visszaterjesztési szabályok származtatása az egyes neuronok definiált relevanciájának (pl. $R_i^{(l)}$) alacsony fokszámú Taylor-polinomos közelítésén alapul. Osztályozási problémák esetén $R^{(L)} = f(x)$, ahol $R^{(L)}$ az utolsó réteg relevanciája $f(x)$ pedig a háló kimenete.

1. ábra
Az ábra a [14] cikkből származik.
Mindkét képhármas balról jobbra: eredeti kép,a pixelenkénti relevancia és a bemeneti kép kiemelt éleinek a szuperpozíciója, végül az eredeti kép és a binarizált relevanciák szuperpozíciója.

Az LRP eredményét az *1. ábra* szemlélteti. A módszer előnye, hogy direkt alkalmazható bármilyen neurális hálóra, nem kell azon változtatni, tovább vagy újratanítani. Hátránya, hogy a relevancia visszaterjesztési szabályok származtatása során jelentős relaxációkat alkalmaz. A magyarázatgeneráló módszerek objektív összehasonlítása nehéz feladat, mivel nem léteznek jól definiált kvantitatív minősítők. Kvalitatív taksálás alapján viszont ez jelenleg a state-of-the-art eljárás.

2.2 Tudás desztillálás

2.2.1 Felhasznált fogalmak

Indokolt a következő módszer alaposabb bemutatása, mivel munkám során ezt is használtam. Ehhez szeretnék néhány olyan fogalmat tisztázni, vagy elnevezni, amit körülményes lenne a módszer bemutatás közben. Ezeket a szavakat a dolgozatom során így fogom használni.

target: A felügyelt tanítás során a tanítóhalmaz összeállítása után (pl. fényképek gyűjtése), minden adatpontról (képről) megállapítjuk, hogy melyik osztályba tartozik. Hibaeloszlásokból származtatott büntetőfüggvények esetén (mint amilyen pl. a kereszt entrópia is) ez jellemzően one-hot kódolt n elemű vektor hozzárendelését jelenti elvárt kimenetként, ahol n az osztályok száma. Például a j. oszály esetén $\boldsymbol{e}_j$.

Ezt az adatponthoz hozzárendelt vektort angol nevén magyarul is legtöbbször „target”-nek, vagyis célnak nevezzük. A felügyelt tanítás során a célunk, hogy a modell ezeket a targeteket minél kisebb hibával megközelítse. Az empirikus kockázat-minimalizálás során ezt a tanító pontokra direkt optimalizációval érjük el, viszont fontos kritérium, hogy a tanítás során nem látott képekre is helyes eredményt adjon (általánosító képesség).

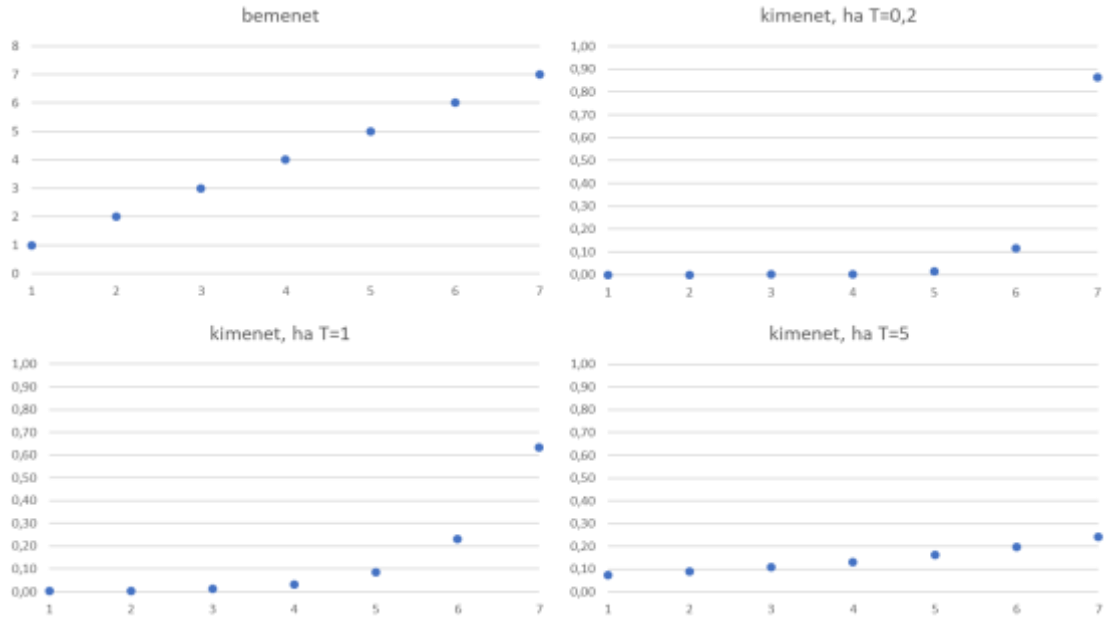
háló struktúra: A neurális hálók struktúrája alatt azon paramétereket értem, melyek nem változnak a tanítás során.

softmax-with-temperature: A softmax gyakran használt aktivációs réteg. Leggyakrabban osztályozási problémák utolsó rétegének aktivációjaként használt, mivel a kimeneti vektor a különböző osztályokba tartozást leíró valószínűségi változó eloszlásaként értelmezhető (ld. logisztikus regresszió). Aktivációs függvényről lévén szó a bemeneti és kimeneti vektorok elemszáma megegyezik. Az aktivációhoz használt Boltzman-eloszlás definíciója:

$$\boldsymbol{q}_i = \frac{e^{\mathbf{z_i}/_T}}{\sum_j e^{\mathbf{z_j}/_T}} \tag{1}$$

Ahol z a bemeneti, q a kimeneti vektor, T pedig a temperature (hőmérséklet tényező). Leggyakoribb esetben a $T = 1$ helyettesítést használjuk. Kisebb T érték választása (a bemenet változatlanul hagyása mellett) nagyobb szórású kimeneti vektort eredményez, ugyanígy nagyobb érték választása kisebb szórásút. A kimeneti vektor elemei között azonban a kisebb-nagyobb relációk nem változnak, így maximum kiválasztás esetén a háló válasza sem.

A softmax működését különböző hőmérsékletekkel az *2. ábrá*n szemléltetem. Látható, hogy a kimenetek között az elemek kisebb-nagyobb relációjában és átlagukban nincsen különbség, csak a szórásuk változik.



2. ábra
A softmax aktiváció kimenete a választott hőmérséklet (T) függvényében.
*A bal felső ábrán egy minta **z** vektor látható. A többi ábrán a softmax kimenete különböző hőmérsékleteken.*

2.2.2 Módszer bemutatása

A tudás desztillálás, vagy angol nevén Knowledge Distillation (KD) eredeti ötlete a szakértő együttesek alkalmazásán alapszik. Hiszen ha egy osztályozási problémához megfelelően adaptálható neurális hálót többször, többféle (random) inicializálás után újratanítunk, akkor az így kapott modellek predikcióit átlagolva az adekvát osztályozás valószínűsége nagyobb, mint csak egy szakértő válaszának használata esetén. Ennek magyarázata, hogy így egy különböző, de pontos szakértőkből felépülő együttest kapunk. Ez a módszer természetesen egy nagyon leegyszerűsített alkalmazása a [12]-ben részletesen tárgyalt szakértő-együttes technikáinak. Azonban most nem lesz szükségünk arra, hogy különböző struktúrájú modellek döntéseit hogyan súlyozzuk az eredmény kialakításában, ugyanis azonos struktúráknál maradunk, melyek válaszait átlagolással aggreáljuk (implicite azt feltételezve, hogy az így előállított szakértők pontossága közel azonos).

A KD lényege, hogy a fenti tapasztalat alapján a sokszereplős döntéshozást próbáljuk meg egy hálónak megtanítani. Az [1] módszere a következő:

$$\boldsymbol{W}^{(i)} = train(X, f^{T=1}(\cdot,\cdot)) \tag{2}$$

ahol $train(\cdot,\cdot)$ a tanító operátor, melynek első argumentuma a tanító pontoknak és a hozzájuk tartozó elvárt kimeneteknek a halmaza (X), második argumentuma pedig a tanított neurális háló paraméteres leképezése. $f^{T}(\cdot,\cdot)$ jelöli a T hőmérsékletű softmax aktivációt alkalmazó neurális hálót. Első paramétere a bemeneti minta, második paramétere a háló súlyai ($\boldsymbol{W}$).

$$\boldsymbol{d'_j} = \frac{1}{2N}\sum_{\boldsymbol{i=1}}^{N} f^{T=t}(\boldsymbol{x}_j, \boldsymbol{W^{(i)}}) + \frac{1}{2}\,\boldsymbol{d_j} \tag{3}$$

$\boldsymbol{d'_j}$ az $\boldsymbol{x}_j$ bemeneti vektorhoz rendelt *soft-target*, $\boldsymbol{d}_j$ ugyanezen bemeneti mintához tartozó *hard-target*, N a szakértők száma, t pedig az alkalmazott hőmérséklet.

$$\boldsymbol{W}^{*} = train(X', f^{T=t}(\cdot,\cdot)) \tag{4}$$

ahol a X' soft-targetekkel módosított tanító halmaz.

A desztillált háló válasza:

$$y(\boldsymbol{x}) = f^{T=1}(\boldsymbol{x}, \boldsymbol{W}^{*}) \tag{5}$$

A már hivatkozott [1]-ben végzett analízis alapján a módszer hatásosan alkalmazható a konstruált eszköz komplexitásának redukciójára minimális pontosságcsökkenés mellett. Nevezetesen a közölt értékek szerint a kiindulási modell 58.9%, míg a 10 db ilyenből álló szakértői együttes 61.1% pontossággal volt képes megjósolni a helyes osztályokat. A szakértői együttes tudását a kiindulási modellbe desztillálva pedig 60.8% pontosságot mértek, ami alig marad el a szakértői együttesétől.

A magas hőmérséklet használata a soft-targetek gyártása és az ezt követő tanítás során azért fontos, hogy kiemeljük a hálók tudását, amit nem kódolnak az eredeti hard-targetek. Abban ugyanis nem különböznek a soft-targetek a hard-targetektől, hogy melyik kimeneten van a legmagasabb érték. Ez az egyes szakértők válaszától függetlenül is mindig teljesül

a (3) összefüggés miatt. Amiben fontosabb különbség mutatkozik, az pont a többi érték. Ezek kisebb-nagyobb relációi azt fejezik ki, hogy az n db hálóból álló rendszer melyik oszályba sorolta volna második, harmadaik, stb… helyen a képet. Soft-targeteket használva tehát nem csak azt tanulja a háló, hogy az adott képet melyik osztályba kellene sorolnia, hanem azt is, hogy melyekre hasonlít egy nagyon sok paraméterű, nagy pontosságú rendszer szerint. Ez által pedig implicite a többi osztályról is tudást szerez.

2.2.3 Felhasználási példa (Soft Decision Trees)

Azt már láttuk, hogy a KD hogyan használható egy adott struktúrájú háló pontosságának növelésére a paraméterszám növekedése nélkül, arra pedig már több módszert is láttunk, hogy hogyan lehet kis pontosságvesztés mellett csökkenteni egy háló erőforrásigényét. A [2] egy olyan módszert tárgyal, amivel KD-t használva a CNN-eknél sokkal átláthatóbb, egészen más típusú modellt tanítunk.

A Soft Decision Tree (enyhe döntési fa), a döntési fáknak egy olyan általánosabb modellje, melyben nem történik minden elágazásnál egyértelmű döntés, hogy merre kell menni, hanem minden lehetséges részfához hozzárendeljük az adott döntés valószínűségét (egy „enyhe döntést” hozunk). Ennek következtében a levelek közül nem pontosan egy adja a minta taksálását, hanem az osztályokba sorolás a levelekig kiszámolt valószínűségekből aggregálódik. [3].

A Soft Decision Tree rendelkezik a döntési fák átláthatóságával, ugyanis minden elágazásról (döntési helyzetről) eldönthető, hogy ott miről dönt éppen a modell. E közben rendelkezik a neurális hálók azon tulajdonságával is, hogy a végső döntést egy valószínűségi eloszlás alapján hozza meg.

A [2] alapján, ezeket a tulajdonságokat felhasználva egy konvolúciós neurális háló tudása desztillálható Soft Decision Tree-be is. Az említett cikkben egy az MNIST adathalmazon 94.45%-os pontosságú Soft Decision Tree-t, és egy 99.21%-os pontosságot elérő CNN-t vettek alapul. A tudásdesztillálással tanított döntési fa pontossága elérte a 96.76%-ot. Így pontosabb modellt kapunk, mint az eredeti döntési fa és átláthatóbbat, mint az eredeti CNN.

3 Klasszikus paraméterredukció desztillálással

Magasabb pontosságot nem csak szakértőegyüttesek alkalmazásával tudunk elérni, hanem nagyobb, így például mélyebb (több rétegű), szélesebb (rétegenként több neuront tartalmazó) vagy bonyolultabban kapcsolt rétegekből álló hálókkal is. Felmerülhet a kérdés, hogy van-e esélyünk egy nagy és komplex háló tudását egy kisebb, így kevesebb paraméterrel rendelkező, gyorsabban futtatható, jobban átlátható, egyszóval hatékonyabb hálónak átadni.

Ennek értelme az lenne, hogy a probléma megoldására használhatnánk jól ismert és tesztelt, jó általánosítóképességű hálót, élesben pedig egy olyat, ami kis méretének következtében hatékonyan oldja meg a feladatot. A kettő problémát szétválasztva pedig oda jutunk, hogy a prototípus megoldás konstrukciója során nem kell azzal foglalkoznunk, hogy hatékonyan oldjuk meg a problémát. Erre elég akkor figyelni, amikor az éles rendszerbe kerülő hálót tervezzük. E hálónak két, egymásnak ellentmondó kritériumnak kell eleget tennie:

1. a nagy háló tudását még képes elfogadhatóan kis hibával modellezni,
2. maximálisan hatékony, tehát működése során a lehető legkevesebb erőforrást használja fel.

InceptionV3

Méréseim alapján az 50 ezer tanítóképből és 10-10 ezer validáló és teszt képből álló ClutteredMNIST adathalmaz problémájára a InceptionV3 [15] nevű, széles körben ismert konvolúciós neurális hálónak nem esik nehezére szinte tökéletesen rátanulnia.

A tanításhoz a fine tuning technikát alkalmaztam, tehát egy más problémára betanult hálót úgy használtam fel, hogy annak utolsó néhány rétegét eldobtam és a helyükre saját, probléma specifikus részhálót konstruáltam, majd az egész hálót tanítottam. Ebben az esetben az 1000 osztályos imagenet problémára tanított hálót vettem a 10 osztályos számjegyfelismerési probléma alapjaként. Ehhez a Tensorflow és a Keras teljeskörű támogatást nyújt.

A fine tuning eredményeképpen a háló a tanító adatokon 99.54%, a validáló adatokon $98{,}58\%$, a teszt adatokon $98{,}29\%$ pontosságot ért el. A tanítás során minden réteg tanult, így több, mint 23 millió tanítandó paraméter volt a rendszerben. Ezeket a pontosságértékeket és a háló a paraméterszámát a dolgozat további részében viszonyítási alapként kezelem.

Tudás desztillálás példa

Az M_1 háló struktúrája legyen a következő:

1. réteg: 2D konvolúció (ablakméret: $3x3$, kimeneti csatornák száma: 32, aktiváció: relu)
2. réteg: maxpooling, (ablakméret: $2x2$)
3. réteg: 2D konvolúció (ablakméret: $3x3$, kimeneti csatornák száma: 32, aktiváció: relu)
4. réteg: maxpooling, (ablakméret: $2x2$)
5. réteg: 2D konvolúció (ablakméret: $3x3$, kimeneti csatornák száma: 32, aktiváció: relu)
6. réteg: maxpooling, (ablakméret: $2x2$)
7. réteg: flatten
8. réteg: fully connected (neuronok száma: 10, aktiváció: softmax)

A softmax hőmérséklettel való használatához a Keras nem nyújt alapértelmezetten támogatást, de ez szükség esetén könnyen megoldható, ha az (1) összefüggést két lépésre bontjuk a következőképpen:

$$\boldsymbol{z}' = {\boldsymbol{z}}/{T}$$
$$\boldsymbol{q}_i = \frac{e^{\boldsymbol{z}'_i}}{\sum_j e^{\boldsymbol{z}'_j}} \tag{6}$$

így a második lépést elvégzi a beépített softmax, az első lépés pedig implementálható Template rétegként.

Az M_1 hálót szintén a ClutteredMNIST adathalmazon tanítva a konstruált hálók pontosságát (tudásdesztillálással és a nélkül) az *1.táblázat*ban foglalom össze:

	M_1	$M_1, KD\ T = 5$	InceptionV3
tanító	60,43%	78,52%	99.54%
validáló	58,42%	74,77%	98,58%
tesztelő	58,43%	74,46%	98,29%

1. táblázat
Az M_1 háló pontosságainak értéke, tudás desztillálással (középen) és a nélkül (bal) a tanító hálóhoz viszonyítva (jobb)

M_1háló tehát, ha nem is képes a maga 39 ezer paraméterével tökéletesen leutánozni az InceptionV3 teljesítményét, de láthatóan sokkal jobb eredményt tudunk elérni, ha a tanításhoz tudásdesztillálást használunk.

4 Redundáns szűrők eliminálása

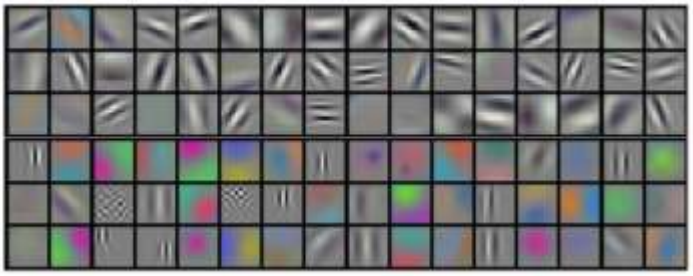
A továbbiakban olyan tulajdonságait tárgyalom a konvolúciós neurális hálóknak, amelyek két megvizsgálandó szempont alapján hatékonyságcsökkentést okoznak, és ezek kiváltására próbálok meg megoldást tanácsolni. Ez a két szempont a következő:

1. megnövekedett erőforrásigényhez vezető tulajdonságok,
2. a háló döntésének magyarázhatóságát rontó tulajdonságok.

Az első vizsgálati szempont magától értetődő, hiszen az minden informatikai rendszer célja, hogy erőforrásainkat minél hatékonyabban kihasználjuk, ezáltal csökkentve a számítási idő, memória és egyéb erőforrások igényét.

A második szempont absztraktabb, ugyanis (a tudomány jelenlegi állása szerint) kvalitatívan nem mérhető. A komplexitás növekedése alapvető velejárója a neurális paradigmának. Minél mélyebb és ez által komplexebb egy neurális háló, annál bonyolultabb leképezések (minták alapján történő) megtanulására alkalmas, melyeket a hagyományos matematikai modellekkel a szakértők nem voltak képesek feltárni. Ez maga után vonja a konstruált háló döntésének nehezebb magyarázhatóságát. Éppen ezért a komplexitás növelés nem célunk, maximum elfogadjuk, ha nem látunk módot hasonló eredmény elérésére kevésbé összetett módszerek alkalmazásával.

Az AlexNet [13] volt az első mély, képfelismerési problémákhoz konstruált konvolúciós neurális háló. Bár az azóta megjelent megoldások (VGG, GoogleNet, ResNet, FractalNet, stb.) nagyobb pontosságú megoldásokat, esetleg hatékonyabb erőforrás kihasználást biztosítanak, mind a mai napig ez a szakirodalomban az egyik legtöbbet vizsgált [16, 17, 18] CNN. Ez annak köszönhető, hogy publikálása fontos szerepet játszott a neurális paradigma újra élesztésében és széles körű elterjedésében. Ahogy a háló alap publikációjának [13] szerzői is írják cikkükben, az áttöréshez szükség volt a GPU-k fejlődésére, az ezekre optimalizált hatékony 2D konvolúció implementációkra, valamint olyan nagy és diverz adathalmazokra, mint az ImageNet.



3. ábra
Az ábra a [13] cikkből származik annak 6. oldalán 'Figure 3' néven található
AlexNet első konvolúciós rétegének 96 db 11x11x3-mas konvolóciós kernele látható rajta.

Az AlexNet első rétegének konvolúciós szűrőit vizualizálja a *3. ábra* (forrás [13]). Könnyen észrevehető, hogy az ábra felső három sorában lévő kernelek érzéketlenek a színekre, míg az alsó háromban lévők többnyire nem. Két másik szembetűnő tulajdonsága is felfedezhető

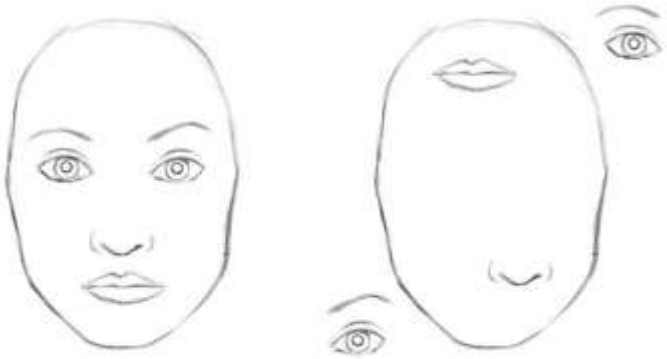
a taglalt filtereknek, melyek általánosan is megfigyelhetőek a CNN-ekben. Egyrészt látható, hogy vannak a megtanultak között szinte minden elemében nulla (a képen teljesen egyenletesen szürke) kernelek. Másrészt vannak olyan kernelek, melyek egymástól szinte egyáltalán nem különböznek.

A szinte csupa nullákból álló szűrés pedig azt jelenti, hogy itt a hálónak nem sikerült semmi értelmes jellemző kiemelését megtanulnia. Ugyanis egy csupa nullásokból álló kernellel való konvolúciónak az eredménye szintén csupa nullákból álló kimenet lesz. Ez pedig az előzőhöz hasonló megfontolásból pazarló, hiszen ha ezeket a paramétereket is sikerült volna rendesen beállítani, akkor talán nőhetne a háló pontossága. Az ilyen szűrések jelentőségével a *4.2* alfejezetben külön foglalkozom.

4.1 Geometriai torzítások

Általános tapasztalat a konvolúciós neurális hálókkal kapcsolatban, hogy bizonyos, az emberi döntéshozatalt egyáltalán nem befolyásoló, bemeneti képeket terhelő változtatásokra a háló nem az elvárt módon reagál. Itt most nem a szintén nagy problémát jelentő és a szakirodalomban is részletesen tárgyalt Adversial Attack [5] problémára utalok, hanem olyan módosításokra, mint a megvilágítás, a képen lévő tárgy színének megváltoztatása, eltolások, forgatások, átskálázás. Míg egy ember számára egyformán nehéz megmondani, hogy milyen fajtájú kutya látható egy képen, akár természetes, akár mesterséges fényben készült az, és ugyanígy, ha a kép egészét, vagy csak negyedét foglalja el. Ugyanezt a viselkedést a CNN-ektől is joggal várjuk el. Ez elérhető, ha megfelelően diverz mintahalmazzal történik a hálók konstruálása, melynek egyik lehetséges módja a minták augmentációja. Általános tapasztalat, hogy az egyes objektumok előfordulásának tanítómintákban látható sokféleségét a hálók redundáns szűrések megtanulásával kezelik [13]. Bár a jelenség robosztusság szempontjából akár előnyös is lehet (pl. Dropout esete), viszont mind interpretálhatóság, mind számítási erőforrásigény szempontjából előnytelen, hiszen az egymással pl. kis mértékű elforgatás, skálázás miatt jelentősen korreláló szűrőkkel történő konvolúciók eredménye is jelentősen korrelál. Ez a jelenség megfigyelhető az AlexNet szűrőiben is (*3. ábra*).

Többek között ezzel a kérdéssel is foglalkozik a [22], melyben egy teljesen új típusú neurális háló struktúrát vezetnek be. A cikk által javasolt kapszula háló ismertetése a dolgozat szempontjából kevésbé fontos, mint motivációinak bemutatása.



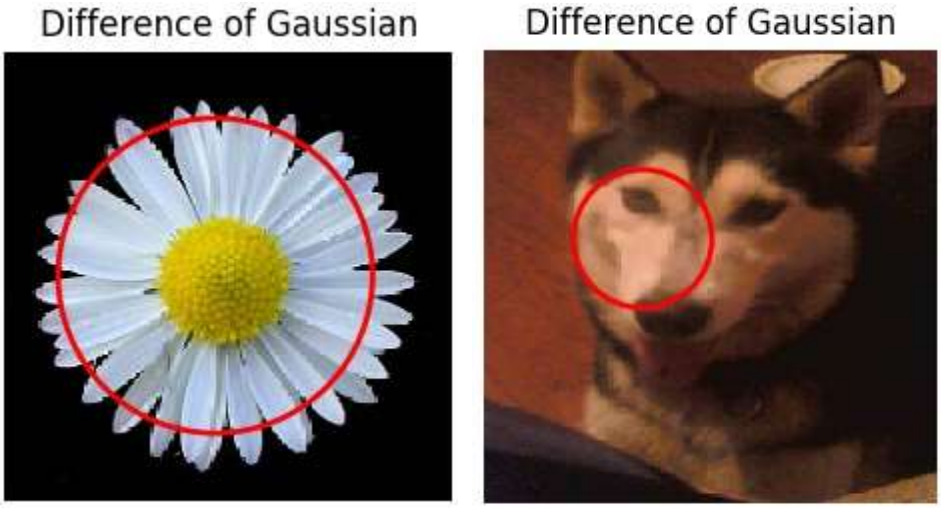
4. ábra
Egy konvolúciós neurális háló számára a két kép hasonló, mivel hasonló részleteket tartalmaznak. (forrás [19])

A kapszula hálók konstrukciójának főbb motivációja az invariáns viselkedés ekvivariánsra cserélése. Ez alatt azt kell érteni, hogy egy komplexebb objektumot (például emberi arcot) felépítő egyszerűbb objektumok megjelenésének állapota (pl. skála, orientáció stb.) között ekvivalencia kell, hogy fennálljon annak érdekében, hogy valóban a kompozit objektumot lássuk. Ennek ellenőrzését a klasszikus, redundáns szűréseket és azokat követő maxpooling-ot tartalmazó hálózatok nem teszik lehetővé (kiváltképp azért nem, mert ritkák azok
a tanítóminták, melyek e miatt okoznának fals osztályozást). A probléma viszont nem jelentkezne, ha a detektálandó objektumunk uniformizált állapotban jelenne meg a háló bemenetén (ez alatt egy regisztrált állapotot értve), mely hatására az egyes rész objektumok is egymással konzisztens állapotban jelennének meg. Ilyen értelemben javulna a háló interpretálhatósága, csökkenne a számítási erőforrásigénye, valamint a robosztussága is javulna.

4.1.1 Uniformizálás

A fentiekben ismertetett problémákra alternatív megoldásként kínálkozik a bemeneti képek uniformizálása. Klasszikus, szakértői képfeldolgozó rendszerek esetén is szinte mindig szerepel az első lépések között a képek valamilyen értelemben uniform állapotra hozása. Ez legtágabb értelemben véve egy olyan előfeldolgozása a bemeneti képeknek, mely során a hasonló objektumokhoz hasonló megjelenési állapotot rendelünk (ez értelmezhető akár *a priori* tudás hozzáadásaként is), ez után adjuk a hálónak a képet feldolgozásra. Konkrét példaképpen pedig előre megpróbáljuk meghatározni a felismerendő objektum méretét, orientációját és csak ez után kerül a kép a hálóhoz, ami megpróbálja felismerni.

Elsőként, csak az objektumok méretének problémájával foglalkozva, megpróbáltam bizonyosságot szerezni arról, hogy ha csak hasonló méretű objektumokat tartalmazó képek képezik a CNN bemenetét, akkor kisebb háló is elég ugyanolyan pontosság eléréséhez. A dolog proof-of-concept jellege miatt nem fordítottam hangsúlyt a skálakompenzáció automatizálására.



5. ábra
A DoG függvényt használó algoritmus eredménye egy olyan képen, amin a felismerendő objektum egy blob (bal) és egy olyanon melyen nem (jobb).

A blob detektálás jól ismert algoritmus a klasszikus képfeldolgozásban. A blob definíció szerint egy világos folt egy sötét képen, vagy egy sötét folt egy világoson, aminek megtalálásához az algoritmus különleges szűréseket használ. Ezen szűrések alapjául a Laplacian of Gaussian (LoG), a Difference of Gaussian (DoG) vagy a Determinant of Hessian (DoH) függvény szolgálhat. Bármelyiket választva, azt az origó körül megforgatni, majd mintavételezni kell. Az így kapott szűrőt megfelelően minden iterációban átméretezve megtalálhatók egy képen a blobok. Az ideális esetet szemlélteti a *5.ábra* bal, és egy kevésbé ideálisat a jobb oldala.

A legegyszerűbb konvolúciós háló, amit megvizsgáltam két konvolúciós rétegből állt. Egyenként 3 illetve 4 kimeneti csatornával, $9x9$-es szűrésekkel. Mindkét konvolúciós réteget relu aktiváció és 2D averagepooling követett. A flatten réteg után egy 10 neuronos fully connected réteg, relu aktivációval, majd a két neuronos kimeneti réteg softmax aktivációval.

Az elért pontosságok: $97{,}3\%$ a tanító, $82{,}3\%$ a validáló és $75{,}0\%$ a teszt adatokon. Ami azt jelenti, hogy a 80 teszt kép közül 60-at felismert a háló.

Ez az eredmény arra engedett következtetni, hogy a képek skálafüggetlenítése mindenképp jó irány, ha hatékony osztályozási feladatokat végző CNN-eket akarunk konstruálni.

4.1.2 Uniformizálás neurális hálóval

A Spatial Transformer Network (STN) [4] egy olyan konvolúciós neurális hálózat, ami regresszióval becsli a merev regisztráció paramétereit. Ehhez tanulás során felhasználja a minták cimkézését ami alapján tanulja a regisztáció paramétereit, mely értékeket direktben a tanítás során nem kap meg. Ez alapján vizsgálódásom szempontjából több előnyös tulajdonsággal is rendelkezik. Először is a regisztráció a klasszikus képfeldolgoásban az a lépés ami a különböző nézőpontokból készült képeket egy közös koordinátarendszerbe transzformálja. A közös koordinátarendszer pedig eltolás, skálázás és forgatás tekintetében uniformizálás. Másodszor, mivel ezt az STN cimkézés alapján tanuja, a képen található objektum alapján különbözően lesznek regisztrálva a képek, a targettől függően. Továbbá, mivel teljesen differenciálható leképezést megvalósító CNN alapú a technológia, ennek köszönhetően önálló rétegként beilleszthető bármely CNN elejére, vagy akár két réteg közé, így az uniformiázáls tanulása együtt történhet az osztályozási probléma tanulásával.

Az STN három fő részből áll:

- egy regressziós modell, ami a képen elvégzendő affin transzfomáció 6 paraméterét becsli
- egy mintavételező rács, aminek segítségével az affin transzformációt elvégezzük a képen,
- és végül egy differenciálható mintavételezési lépés.

A mintavételezésnek, azért kell differenciálhatónak lennie, hogy a kimeneti hibája is visszaterjeszthető legyen, egy felelől az STN regressziós modelljének, másfelől, ha az STN előtt is rétegeknek.

A regressziós modell lehet bármilyen neurális háló alapú megoldás. Az általam használt (a továbbiakban STN rétegként hivatkozott) modell a következő volt:

1. réteg: maxpooling (ablakméret: $2x2$)
2. réteg: 2D konvolúció (ablakméret: $5x5$, kimeneti csatornák száma: 20)
3. réteg: maxpooling (ablakméret: $2x2$)
4. réteg: 2D konvolúció (ablakméret: $5x5$, kimeneti csatornák száma: 20)
5. réteg: flatten
6. réteg: fully connected (neuronok száma: 50, aktiváció: relu)
7. réteg: fully connected (neuronok száma: 6)

A regressziós modellt bilineáris interpoláció követi. Így ha egy rétegként tekintünk az előbb részletezett 7 rétegre és az interpolációra, akkor egy olyan réteget kapunk, ami a bemeneti képen egy affin transzformációt és egy mintavételezést végez a réteg kimeneti hibájának minimalizálása érdekében.

Méréseim során az STN minden esetben a legelső rétege volt a hálónak. Ennek magyarázata, hogy így az STN kiementét bementként kapó konvolúciós szűrések már nem kell, hogy feleslegesen redundánsak legyenek. A ClutteredMNIST adathalmazból származó $60x60$ pixeles bemeneti képen minden esetben $30x30$-as mintavételező rácsot alkalmaztam, bilineáris intepolációval. A továbbiakban a regressziós modell hatékonyságának növelése nem volt cél, mivel paraméterszámban nem jelentős, és a kimenete könnyen értelmezhető, így interpretálhatóság szempontjából is kvalitatíve hatékonynak minősül.

4.1.3 Tapasztalati eredmények

Elsőként az alábbi háló architektúrát vizsgáltam:

0. réteg: STN réteg (kimeneti kép: $30x30$, bilineáris interpoláció)
1. réteg: 2D konvolúció (ablakméret: $3x3$, kimeneti csatornák száma: 32, aktiváció: relu)
2. réteg: maxpooling, (ablakméret: $2x2$)
3. réteg: 2D konvolúció (ablakméret: $3x3$, kimeneti csatornák száma: 32, aktiváció: relu)
4. réteg: maxpooling, (ablakméret: $2x2$)
5. réteg: flatten
6. réteg: fully connected (neuronok száma: 256, aktiváció: relu)
7. réteg: fully connected (neuronok száma: 10, aktiváció: softmax)

A ClutteredMNIST adathalmzon a tanítás után kapott pontosság értékek a tanító képeken $97{,}66\%$, a validáló és a teszt képeken pedig $95{,}4\%$. Figyelembe véve, hogy ugyanezen probléma esetén az Inception V3 alapú megoldás pontossága mindkét esetben kevesebb, mint 3%-al haladja meg a vizsgált háló pontosságát, míg a paraméterek számában nagyságrendileg 50-szeres az eltérés, bizonyossá vált az uniformizálás hatásossága.

Első észrevételem, hogy előzetes várakozásaimmal ellentétben az InceptionV3 tudásának desztillálásával nem kaptam szignifikánsan pontosabb modellt. Ehhez persze az is hozzájárul, hogy a modell már eleve nagyon pontos és szignifikáns különbség kimérésére nincs is érdemben lehetőség. Az ezzel kapcsolatos méréseim eredményét a *2.táblázat*ban foglalom össze.

	hard target	soft target T=1	s.t. T=2	s.t. T=5	s.t. T=10	Incept.V3
tanító	$\mathbf{97,66}\%$	97,30%	97,09%	98,41%	98,01%	99.54%
validáló	$\mathbf{95,40}\%$	94,89%	95,14%	95,66%	94,75%	98,58%
tesztelő	$\mathbf{95,41}\%$	95,07%	95,06%	95,14%	94,69%	98,29%

2. táblázat

A vizsgált háló pontossága a cluttered MNIST adathalmazon különböző hőmérséklet értékek használatával.

A továbbiakban az STN réteg hatákonyságával kapcsolatban végeztem méréseket, miközben az utána következő rétegek architektúráját módosítottam. Először is a 6. réteget elhagytam. Ez hatékonyság szempontjából fontos lépés lehet, úgyanis egy részről ennek a rétegnek az elhagyásával az háló paramétereinek a száma 399.202-ről 112.994-re csökkent, másrészről pedig minél kevesebb fully connetced réteget tartalmaz egy neurális háló, annál egyszerűbb a döntéseinek magyarázata.

Az ezzel a módoítással végzett méréseim eredménye a *3.táblázat*ban látható. Az első oszlopból leolvasható, hogy a réteg elhagyásával a háló pontossága nem romlott drasztikusan. Validáló és tesztelő képek esetében valamivel több, mint 2% a romlás. A tanító adatokon ennél kicsit több, de ez azzal magyarázható, hogy a fully connected réteg több paramétere révén hajlamosabbá teszi a hálót a túltanulásra, így ez a kicsit nagyobb százalékos romlás nem írható egyértelműen a veszteségek közé.

	hard target	soft target T=1	s.t. T=2	s.t. T=5	s.t. T=10	Incept.V3
tanító	$\mathbf{94,45}\%$	94,53%	95,29%	94,44%	93,96%	99.54%
validáló	$\mathbf{93,17}\%$	92,94%	93,28%	91,18%	90,43%	98,58%
tesztelő	$\mathbf{93,01}\%$	93,05%	92,82%	91,53%	90,57%	98,29%

3. táblázat

Az utolsó előtti fully connected layer elhagyásával kapott háló pontossága különböző hőmérséklet értékek használatával.

Ez után az STN utáni maxpooling rétegeket cseréltem averagepooling rétegekre. Ennek a lépésnek a magyarázata a [22] alapján az, hogy a maxpooling rétegek egymás után többszöri alkalmazása érzéketlenné teszi a hálót az objektum részeinek egymáshoz viszonyított helyzetére és méretére, mivel ezek kimenete a régió egyetlen pixelétől függ csak. Ez persze javarészt nem jelentkezik, ha uniformizált a bemenet, hiszen ilyen esetekben a réteg bemenete nem elegendően diverz a nem kívánt invariancia kialakulásához. A csere hatására a háló tanulandó paramétereinek a száma természetesen nem változott.

Ez a változtatás ismét kicsit több, mint 2% romlást okozott a háló validáló és tesztelő pontosságában. Látszik azonban az is, hogy ez a lépés csak a háló általánosítóképességét rontotta, a tanító adatokon a háló az előző mért értéknél még jobban is teljesített.

	hard target	soft target T=1	s.t. T=2	s.t. T=5	s.t. T=10	Incept.V3
tanító	$\mathbf{95,35}\%$	95,29%	$94{,}20\%$	$93{,}41\%$	92,80%	99.54%
validáló	$\mathbf{91,52}\%$	91,11%	$90{,}45\%$	88,65%	87,69%	$98{,}58\%$
tesztelő	$\mathbf{91,28}\%$	$91{,}18\%$	90,34%	88,23%	$88{,}01\%$	$98{,}29\%$

4. táblázat

A maxpoolingok avaragepoolingra cserélésével kapott háló pontossága különböző hőmérséklet értékek használatával.

Az sajnos világosan látszik a 2., 3., 4. táblázatokból, hogy a tudás desztillálás módszere nem csak az első esetben tapasztalt túl közeli eredmény miatt nem működik. Az egyre rosszabb eredmények feltevésem szerint azzal magyarázhatóak, hogy mint ahogy azt fentebb már taglaltam a desztillálás módszerével a háló azt tudja megtanulni egy másik hálótól, hogy mely osztályok mintái mutatnak jelentős hasonlóságot Ennek a tudásnak a kihasználásához viszont az kell, hogy a két háló ugyanazon eloszlásból származó mintákat lásson. Mivel az STN réteg struktúrája miatt az azonos osztályba tartozó minták uniformizálására törekszik, míg ennek megfelelő transzformációkat nem végez az InceptionV3 alapú megoldás, ezért hasznos plussz információt nem tud szolgáltatni a vizsgált esetekben.

Viszont a háló pontossága két nehezen interpretálható rétegnek az eldobása után (melyek közül az egyik paraméterszámban önmagában is nagyobb, volt, mint az így kapott háló) még mindig 90% felett tartható. Ez markánsan alátámasztja a háló növekvő hatékonyságát a képek uniformizálása következtében. Már csak az a kérdés, hogy még kevesebb réteg alkalmazásával megtartható-e a háló válaszának nagy pontossága.

A két $3x3$ konvolúciós réteget megpróbálhatjuk eggyel kiváltani. Melyhez az kell, hogy az új réteg egy pixelének érzékenységi mezője legalább akkora legyen, mint az eddigi két konvolúciós rétegé. Ehhez egy $9x9$ méretű kernel használata elegendő, mivel a második konvolúciós réteg kimenetének egy pixelének az értéke a bemenetének $3x3$ pixelétől függ. Ennek a $3x3$-mas résznek az előállításához az averagepooling egy $6x6$ pixelnyi területet dolgozott fel. Amely pixelek pedig az első konvolúciós réteg kimenetén jelennek meg, úgy, hogy ezek értéke a bemenet $8x8$ pixelétől függ. Mivel általában nem alakalmazunk páros méretű konvolúciós szűrőket, ezért $9x9$ méretű kernelt használtam. Azért, hogy a kimenet se legyen nagyobb dimenziójú, mint a korábbi két réteg kimenete $2x2$ strideot is alkalmaztam. Az így kapott réteg által modellezhető leképezések halmaza nem áll egyértelmű relációban az eredeti két réteg által megvalósítható leképezések halmazával. Ezen kívül a Relu aktivációt LeakyRelura cseréltem, mert ez egy invertálható nem linearitás, ami elősegítheti a háló döntésének interpretációját. Fontos megjegyezni, hogy az által, hogy csak egy konvolúciós réteget alkamazok, az ezen réteg beli szűrők definíciója alapján direkt vizuálisan interpretálhatóvá válik a háló általános döntéshozatala.

Az így konstruált háló tehát :

0. réteg: STN réteg (kimeneti kép: $30x30$, bilineáris interpoláció)
1. réteg: 2D konvolúció (ablakméret: $9x9$, kimeneti csatornák száma: 32, aktiváció: leakyrelu)
2.réteg: averagepooling, (ablakméret: $9x9$)
3.réteg: flatten
4.réteg: fully connected (neuronok száma: 10, aktiváció: softmax)

A paraméterszáma 110.210, ami a kiindulási modell paraméterszámának csaknem negyede. A tanítással elért pontosság pedig a tanító képekre $\mathbf{93,73}\%$, a validáló és teszt képekre pedig $\mathbf{89,14}\%$, illetve $\mathbf{89,53}\%$. Ez az eredmény egy kicsit elmarad a kitűzött 90%-tól, viszont tovább javítható, ha a konvolúciós rétegnek több kimeneti csatornája van. A kívánt pontosság már 47 csatorna használatával előáll, de tovább növelhető, még több csatorna használatával. Azonban tetszőlegesen sok csatornát sem érdemes tetszőlegesen sok csatornát alkalmaznunk, hiszen az egyre több és több csatornának egyre kevesebb a pontossághoz hozzáadott értéke, ez pedig hatékonyságcsökkenéshez vezet. Továbbá ronthatja a háló általánosító képességét is.

Az így konstruált háló pontosságának alakulását a csatornák számának a függvényében az *5. táblázat*ban foglalom össze.

csatorna szám	32	**42**	64	128	256	1024
tanító	93,73%	**96,02%**	96,39%	94,56%	96,71%	96,31%
validáló	89,14%	**90,33%**	90,82%	90,45%	91,61%	92,38%
tesztelő	89,53%	**90,26%**	91,30%	90,54%	92,02%	92,70%
paraméterszám	110,210	**118,790**	128,514	165,122	238,338	677,634

5. táblázat
Az egy konvolúciós réteget tartalmazó háló pontossága a konvolúciós réteg kimeneti csatornaszámának függvényében.

4.1.4 Összefoglalás

Ebben a fejezetben tahát rámutattam az uniformizálás jelentőségére a konvolúciós neurális hálók esetében. Ez után először proof-of-concept jellegel a blobdetektáló algoritmust használva, majd a neurális háló alapú Spatial Transformer Networkot alkalmazéséval be is bizonyítottam, hogy szignifikánsan hatékonyabb hálózatok tervezhetőek ilyen módon. Majd pedig interpretálhatósági megfontolások mentén tovább egyszerűsítettem a háló struktúráját, minek hatására az STN után egyetlen konvolúciós és szintén csupán egy darab fully connected rétegből álló hálóval is sikerült a pontosságot a tanító, validáló és tesztelő adatokon is 90% felett tartani.

Fontosnak tartom még kiemelni, hogy az STN-nek ugyanúgy diverz tanítóadatokra van szüksége, hogy igazán jól alkalmazható legyen. Használatával viszont a hálónak egy a többitől jól elválasztható része foglalkozik a skálázás, eltolás, forgatás kérdésével, így (mérésekkel is alátámasztva) biztosabbak lehetünk benne, hogy ezekből nem származik redundancia. Ennek hatására a háló komplexitása jelentősen csökken, továbbá a döntéshozatal értelmezése is jelentősen egyszerűsödik, hiszen kevesebb és kevésbé korrelált szűrések sorozatának függvénye a bemeneti minta taksálása.

4.2 Háló tanítása L0 regularizációval

Az előző fejezet végére sikerült egy meglehetősen sekély hálózatot létrehozni, melynek hibája bár folyamatosan nőtt az egyszerűsítések mentén, sikerült azt racionális kereteken belül tartani. Arra is rámutattam, hogy a pontosság növelhető, ha szélességét növeljük. Ebben a fejezetben arra teszek ajánlást, hogy hogyan lehet ezt is hatásosan megvalósítani. A javasolt módszer általánosan alkalmazható CNN-ek hatékonyabbá tételére.

4.2.1 Motiváció

Általánosan alkalmazott módszer CNN tervezéskor, hogy a bemenettől távolodva egyre több kimeneti csatornával rendelkezzenek a konvolúciós rétegek (miközben az egyes csatornák felbontása monoton csökken). Az InceptionV3 például az általam használt konstrukciójában 2048db $3x3$ méterű csatornát állít elő a bemeneti képből a döntéshozáshoz, a [6]-ban bemutatott 34 rétegű ResNet pedig 512db $3x3$ méretűt. A sok csatorna azért szükséges, hogy minél többféle szűrésen, jellemző kiemelésen menjen keresztül a kép (ezáltal növelve a háló által approximálható nemlineáris leképezések halmazát). Az előző fejezet végén méréssel is alátámasztottam, hogy minél több csatornából áll az utolsó konvolúciós réteg kimenete, annál nagyobb pontosságot lehet elérni.

Azonban ez a módszer is hatékonyságcsökkenéssel jár mindkét vizsgált szempont szerint. Először is bár a paraméterszám növekedés nem drasztikus (az 5. táblázatban látható konkrét példában 572-vel nő a tanulandó paraméterek száma minden új csatornával) a konvolúciós műveletek elvégzése teszi ki a CNN-ek számítási idejének nagy részét. Az interpretálhatóság pedig az által nehezedik, hogy minden kimeneti neuron az összes csatornától függ.

Indokolt tehát a tanítás közben a hálót arra késztetni, hogy csak a döntéshozás szempontjából szükséges konvolúciókat végezze el és csak az adott neuron szempontjából szükséges csatornákat vegye figyelembe.

A *2.1.2 alfejezetben már röviden* ismertettem paraméter nyesést végző módszereket. A [9]-ben javasolt eljárás súlyonként végez vágást a kimenetre gyakorolt hatást minimalizálva. A [10] algoritmus pedig a hasonló súlymátrixszal rendelkező neuronokat vonja össze. Ezen módszerek hátránya, hogy a tanítás során nem cél a vágás elősegítése.

4.2.2 Regularizáció

Felügyelt tanulás minden esetében egy jól definiált *célfüggvény* minimalizálására törekszünk:

$$arg\min_{\boldsymbol{W}}\{C(\boldsymbol{W})\} \tag{7}$$

C a tanítás során minimalizálandó célfüggvény, $\boldsymbol{W}$ pedig a neurális háló súlyait tartalmazó tenzor, melyet a későbbiekben három változóval indexelünk, és minden eleme egy 2D mátrix ($\boldsymbol{W}_{i,j}^{(l)}$ jelöli az l. réteg i. csatornája és az l. réteg j. csatornája közötti szűrő súlymátrixát). Ez a célfüggvény legegyszerűbb esetben (tisztán frekventista megközelítés) a predikciók *hibájának* összege, ahol a hiba kiszámítása szintén előre meghatározott módon, a targetek függvényében történik:

$$C(\boldsymbol{W}) = \sum_i \ell(f(\boldsymbol{x}_i, \boldsymbol{W}), \boldsymbol{d}_i) \tag{8}$$

ahol ℓ a veszteségfüggvény, $f(\boldsymbol{x}_i, \boldsymbol{W})$ a háló kimenete $\boldsymbol{x}_i$ bemenetre $\boldsymbol{W}$ súlyok mellett, $\boldsymbol{d}_i$ pedig az $\boldsymbol{x}_i$-hez tartozó target.

Regularizáció során a célfüggvényt kiegészítjük egy *büntető függvénnyel* is. Ezért a minimalizálás hatására a $\boldsymbol{W}$ a büntetőfüggvény által impliciten meghatározott tulajdonságokkal fog rendelkezni (és ezzel áttérünk a Bayes-i paraméterbecslésre). Általánosan regularizáció esetén a minimalizálandó összefüggés:

$$C(\boldsymbol{W}) = \sum_i \ell(f(\boldsymbol{x}_i, \boldsymbol{W}), \boldsymbol{d}_i) + \rho \cdot \mathcal{H}(\boldsymbol{W}) \tag{9}$$

ahol ρ hiperparaméter, $\mathcal{H}$ pedig a regularizációs büntetőfüggvény. A továbbiakban az első tagot likelihood tagnak hívom. Először vizsgáljuk meg, hogy mik az ismertebb büntetőfüggvények.

A Tikhonov regularizációnak is nevezett L_2^2 (L_2 norma négyzet):

$$L_2^2 : \mathcal{H}(\boldsymbol{W}) = \sum_{l,i,j} \sum_{u,v} \left| \boldsymbol{W}_{i,j}^{(l)}(u,v) \right|^2 \tag{10}$$

Az összefüggés alapján látható, hogy a célfüggvényhez hozzáadjuk a háló súlyainak négyzetösszegét, ennek hatására a tanítás során arra bíztatjuk a hálót, hogy minél kisebbek legyenek a súlyok. Széleskörűen alkalmazott regularizácó a neurális hálók tanítása során, belátható, hogy a korai leállás, illetve a neuronok kimenetének zajjal történő terhelése is hasonló jellegű regularizáció. Érdemes megjegyezni, hogy a szakzsargonba hibásan ezt nevezik L_2 regularizációnak. Mivel a későbbiekben azon módszert is vizsgálni fogom, ezért én itt nem azt az elnevezést használom.

Az L_1 büntetőfüggvényes regularizáció során a súlyok négyzetösszege helyett az abszolútértékük adódik hozzá a célfüggvényhez.

$$L_1 : \mathcal{H}(\boldsymbol{W}) = \sum_{l,i,j} \sum_{u,v} \left| \boldsymbol{W}_{i,j}^{(l)}(u,v) \right| \tag{11}$$

Így az L_2^2-nél relatíve kevésbé bünteti a nagyobb súlyokat, azonban a kisebbeket jobban. Könnyen belátható, hogy általános esetben ez a ritkasági regularizációt legkisebb hibával közelítő konvex büntetőfüggvény. Alkalmazása során a háló motiváltabb, hogy a kisebb amplitúdójú, ezért irrelevánsnak tűnő súlyokat teljesen kinullázza, miközben kevésbé érzékeny a nagy súlyokra. Szintén gyakran alkalmazott és a DeepLearning keretrendszerek által is támogatott regularizációs típus.

Az L_0 norma alapú ritkasági regularizációt főleg jelfeldolgozásban alkalmazzuk, de létezik gépi tanulásos megfelelője is (pl. az adaptive Lasso [20]):

$$L_0 : \mathcal{H}(\boldsymbol{W}) = \sum_{l,i,j} \sum_{u,v} 1 - \delta\left(\boldsymbol{W}_{i,j}^{(l)}(u,v)\right) \tag{12}$$

δ a jelfeldolgozásból ismert Kronecker delta függvény tehát:

$$\delta(x) = \begin{cases} 1 & ha\ x = 0 \\ 0 & egyébként \end{cases} \tag{13}$$

Az L_0 regularizáció során tehát a célfüggvényhez a $\boldsymbol{W}$ nem nulla elemeinek a számát adjuk hozzá. Ez az L_1-hez hasonló viselkedést eredményez, azt hozzátéve, hogy kicsi és nagy nem 0 elemekre azonos a büntetés, ennek köszönhetően nem érzékeny a súlyok amplitudójára. L_0 regularizációt szokás szerint nem alkalmazunk neurális hálók tanítása során. Ennek oka, hogy egyrészt nem konvex, másrészt minden pontjában, ahol deriválható, 0 a derivált értéke., A széles körűen alkalmazott gradient descent (és az erre épülő klasszikus eljárások, pl. momentumos gradiens, adaptív gradiens módszerek) a C célfüggvény minimalizálása során csak korlátolt mértékben képesek a regularizációs büntetőfüggvény minimalizálására.

A fenti regularizációk mind egyesével büntetik a súlyokat (a hozzájuk kapcsolódó minimalizálási problémák teljesen szeparábilisek), így hatásuk is súlyonként érvényesül. Ezzel szemben a következő regularizációk a súlyok egy csoportját bünteti.

Az L_2 regularizáció mátrixokon értelmezett analógiája:

$$L_F : \mathcal{H}(\boldsymbol{W}) = \sum_{l,i,j} \left\| \boldsymbol{W}_{i,j}^{(l)} \right\|_F \tag{14}$$

a Frobenius-norma definíciója pedig:

$$\left\| \boldsymbol{W}_{i,j}^{(l)} \right\|_F = \sqrt{\sum_{u,v} \left(\boldsymbol{W}_{i,j}^{(l)}(u,v) \right)^2} \tag{15}$$

Tehát így konvolúciós kernelenként adódik hiba a célfüggvényhez, ami a később részletezett okokból előnyös.

A fent ismert regularizációs módszerek után definiáljuk $Tenzor\ L_0$ néven a következő büntető függvényt:

$$Tenzor\ L_0 : \mathcal{H}(\boldsymbol{W}) = \sum_{l,i,j} 1 - \delta\left(\left\| \boldsymbol{W}_{i,j}^{(l)} \right\|_F \right) \tag{16}$$

Ennek az új regularizációnak a hatása az L_F-éhez lesz hasonló abban a tekintetben, hogy súlyok egy csoportját bünteti és L_0-éhoz abban a tekintetben, hogy a nem csupa 0 súlymátrixok száma jelenik meg a büntetőfüggvényben. Ezzel tehát a két viselkedést ötvözve arra sarkalljuk a hálót, hogy a tanítás során minél kevesebb nem 0 csupa súlymátrix keletkezzen.

Három kérdés merül fel ezen a ponton. Az első kérdés, hogy miért akarunk súlyokat csoportonként regularizálni. Erre a következő fejezetben adom meg a választ. A második kérdés, hogy mit érünk egy olyan neurális hálóval, amit a tanítás során rávettünk arra, hogy minden súlymátrixa csupa 0 tagokból álljon. Erre a válasz az, hogy patológiás esetek kivételével nem állhat elő, hogy minden mátrix csupa 0 tagú legyen. Az összes büntető tag csupán, mint plusz megkötés jelentkezik a frekventista (8) összefüggés mellett és a ρ

hiperparaméterrel tudjuk szabályozni, hogy ennek a hatása mennyire markánsan érvényesüljön. Ha tehát egy súly mátrix kinullázása nagyobb hibát okoz a likelihood tagban, mint amennyit javít a büntető tagon, akkor nem fog változni.

A harmadik kérdés, hogy miként tudjuk érvényre juttatni a $Tenzor\ L_0$ regularizáció hatását, ha tudjuk, hogy ennek is, mint az L_0-nak 0 a deriváltja minden pontban (a 0 pontot leszámítva, ahol pedig nem deriválható), tehát a C célfüggvény gradiensében nem jelenik meg ettől függő tag.

Erre megoldást adhat a proximity gradient módszer melynek segítségével iteratívan tudjuk közelíteni a büntető függvénnyel sújtott célfüggvény egy kritikus pontját.

A gradient descent algoritmus a minimumhelyet a következő összefüggés alapján számolja iteratívan:

$$\boldsymbol{W}^{(t+1)} = \boldsymbol{W}^{(t)} - \mu \cdot \nabla C(\boldsymbol{W}) \tag{17}$$

ahol az eddigi jelöléstől eltérően $\boldsymbol{W}^{(t)}$ a mátrix súlyait tartalmazó tenzor, az algoritmus t. iterációja után, μ pedig hiperparaméter, az úgynevezett bátorsági faktor.

Proximal gradiens módszer használata esetén a súlytenzor frissítése két lépésre bontható a következő módon:

$$\boldsymbol{W}' = \boldsymbol{W}^{(t)} - \mu \cdot \nabla_{\boldsymbol{W}} \sum_i \ell(f(\boldsymbol{x}_i, \boldsymbol{W}), \boldsymbol{d}_i)$$
$$\boldsymbol{W}^{(t+1)} = prox_{\rho,\mathcal{H}}(\boldsymbol{W}') \tag{18}$$

Az első lépés tehát azonos a korábban látottal, majd alkalmazzuk a proximity operátort:

$$prox_{\rho,\mathcal{H}}(\boldsymbol{W}') = arg\ \min_{\boldsymbol{W}} \left\{ \frac{1}{\mu} \|\boldsymbol{W} - \boldsymbol{W}'\|_2^2 + \rho\, \mathcal{H}(\boldsymbol{W}) \right\} \tag{19}$$

A korábbi definícióba (15) behelyettesítve:

$$\|\boldsymbol{W} - \boldsymbol{W}'\|_2^2 = \sum_{l,i,j,u,v} \left(\boldsymbol{W}_{i,j}^{(l)}(u,v) - \boldsymbol{W}'^{(l)}_{i,j}(u,v) \right)^2 \tag{20}$$

Ez a tag tehát úgy interpretálható, hogy a $\boldsymbol{W}'$minél kisebb mértékű megváltoztatása mellett szeretnénk érvényre juttatni a büntető tagot, melyet a különböző regularizációk esetén a következőképpen írhatunk fel: (ismét visszatérek arra a jelölésre mikor, a felső index a réteg sorszáma.)

$$L_2^2 : \boldsymbol{W}_{i,j}^{(l)}(u,v) = \boldsymbol{W}'^{(l)}_{i,j}(u,v) \Big/ \left(1 + \frac{\rho}{2\mu}\right) = \boldsymbol{W}'^{(l)}_{i,j}(u,v) \cdot \alpha \tag{21}$$

α hiperparaméter csak ρ-tól függ, így az önálló hiperparaméterként hangolható. Látható, hogy a büntető tag multiplikatívan csökkenteni a súlyok értékét, így nagyobb mértékben redukálja a nagyobb súlyokat.

$$L_1 : \boldsymbol{W}_{i,j}^{(l)}(u,v) \;=\; max\left\{0, \left|\boldsymbol{W'}_{i,j}^{(l)}(u,v)\right| - \frac{\rho}{2\mu}\right\} sign\left\{\boldsymbol{W'}_{i,j}^{(l)}\right\} \tag{22}$$

A súlyokat additívan csökkenti, így a (11) összefüggéssel analóg módon kevésbé érzékeny a nagyobb abszolút értékű súlyokra. Ezt a műveletet hívja a szakzsargon zsugorításnak. A folyamatos csökkentés ismét csak azért nem vezet csupa 0 súlyokkal rendelkező hálóhoz (egyik esetben sem), mert az első lépés hatása erőteljesebb azon súlyok esetén, melyek eldobása túl nagy hibát eredményezne (a likelihood tag hatása miatt megmaradnak).

$$L_0 : \boldsymbol{W}_{i,j}^{(l)}(u,v) \;= \begin{cases} 0 & ha \left|\boldsymbol{W'}_{i,j}^{(l)}(u,v)\right| < \rho\mu \\ \boldsymbol{W'}_{i,j}^{(l)}(u,v) & egyébként \end{cases} \tag{23}$$

Az L_0 regularizáció a küszöbértéknél kisebb súlyokat kinullázza, minden iterációban, tehát a (12) összefüggéssel analóg módon a nem 0 elemek számát csökkenti.

Az L_2 regularizáció proximity operátorában megjelenik a mátrix (vektorok euklideszi hosszával analóg) Frobenius-normája (melyet β jelöl):

$$L_2 : \boldsymbol{W}_{i,j}^{(l)} \;= \begin{cases} \left(\boldsymbol{W'}_{i,j}^{(l+1)} / \beta_{i,j}^{(l+1)}\right) \cdot \beta_{i,j}^{(l+1)} \left(1 - \frac{\rho}{2\mu}\right) & ha\ \beta_{i,j}^{(l+1)} \geq \frac{\rho}{2\mu} \\ \mathbf{0} & egyébként \end{cases} \tag{24}$$

tehát bár kinézetre valóban hasonlít az L_2^2-re, viszont az eljárás csak a súlymátrixok Frobenius-normáját zsugorítja, az egyes súlyok egymáshoz viszonyított értékét nem módosítja. Ezáltal teljes mátrixok kinullázására törekszik az egyes súlyok kinullázása helyett.

Végül a $Tenzor\ L_0$ regularizáció proximity operátorát ismét az L_0 és az L_2 proximity operátorok kombinációjaként kapjuk:

$$Tenzor\ L_0 : \boldsymbol{W}_{i,j}^{(l)} \;= \begin{cases} \mathbf{0} & ha\ \left\|\boldsymbol{W'}_{i,j}^{(l+1)}\right\|_F < \rho\mu \\ \boldsymbol{W'}_{i,j}^{(l)} & egyébként \end{cases} \tag{25}$$

Tehát ha a mátrix Frobenius-normája a küszöb alatt van, akkor az egész mátrixot kinullázza, ellenkező esetben nem módosítja. Az összefüggésből látható, hogy itt nincs semmilyen zsugorító hatás, ezért bár mérsékelten, de megmarad ugyanaz a probléma, mint ami a gradient descent alapú L_0 minimalizációnál jelentkezett. Ezen az iteratívan újrasúlyozott L_2 norma alapú megoldás segíthet [21], mely az L_0 normás minimalizációt iteratívan újrasúlyozott, L_2 normás minimalizációk sorozatára vezeti vissza:

$$IRWL_2 : \lim_{t\to\infty}\left(\boldsymbol{W}_{i,j}^{(l,t)} = argmin_{\boldsymbol{W}}\left\{\frac{1}{\mu}\left\|\boldsymbol{W} - \boldsymbol{W'}_{i,j}^{(l)}\right\|_2^2 + \rho\frac{\|\boldsymbol{W}\|_{Frobenius}}{\left\|\boldsymbol{W}_{i,j}^{(l,t-1)}\right\|_{Frobenius} + \varepsilon}\right\}\right) \tag{26}$$

4.2.3 Súlyok regularizálása csoportosan

Fully connceted neurális háló esetén az $l+1$-edik rétegében egy neuron aktivációját az alábbi összefüggés definiálja:

$$\boldsymbol{L}_j^{(l+1)} = \sum_k \boldsymbol{W}_{j,k}^{(l+1)} \cdot \boldsymbol{L}_k^l \tag{27}$$

Ahol L^l az l-edik réteg neuronjainak vektora, $\boldsymbol{W}^l$ a két réteg közötti súlymátrix.

Konvolúciós rétegek esetén az $\mathrm{l}+1.$ réteg egy pixelének aktivációját definiálja a következő kifejezés:

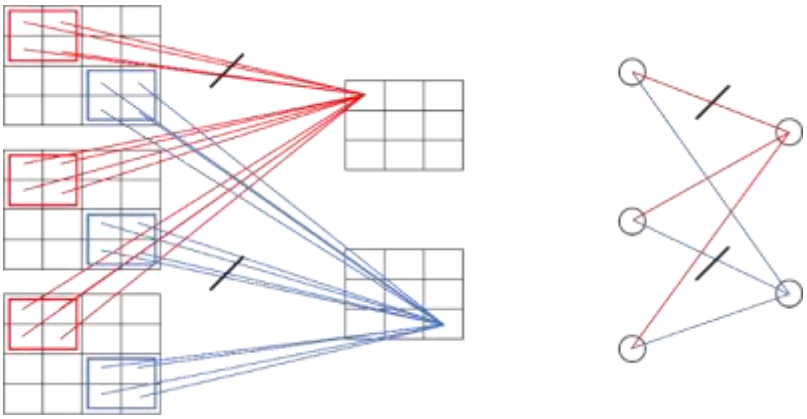
$$L_i^{l+1}(u,v) = \sum_j \left(\sum_{u',v'} \boldsymbol{W}_{i,j}^{(l+1)}(u',v') \cdot \boldsymbol{L}_j^i(u-u', v-v') \right) \tag{28}$$

ahol L_j^l a $l.$ réteg $j.$ csatornája, $\boldsymbol{W}_{i,j}^l$pedig egy NxN méretű konvolúciós szűrő ablak. Mindkét hálótípus esetében elterjedten alkalmazzák $\boldsymbol{W}$elemeire az L_1 és L_2^2 regularizációkat, melyek hatását korábban már ismertettem. Azonban CNN-ek esetén érdemes a 28) összefüggést a következő alakra hozni és úgy is megvizsgálni:

$$\boldsymbol{L}_i^{(l+1)} = \sum_j \boldsymbol{W}_{i,j}^{(l+1)} * \boldsymbol{L}_j^i \tag{29}$$

ahol az $\boldsymbol{L}^l$ az $l.$ réteg csatornáit, a $\boldsymbol{W}_{i,j}^l$ pedig a két csatorna közti konvolúciós kernel.

A konvolúciós hálók csatornái interpretálhatók a fully connected háló neuronjainak több dimenziós bemeneti jelet váró kiterjesztéseként. Mivel egy réteg csatornáit úgy kapjuk, hogy az előző réteg csatornáin lineáris szűrést végzünk és az eredményeket szuperponáljuk.

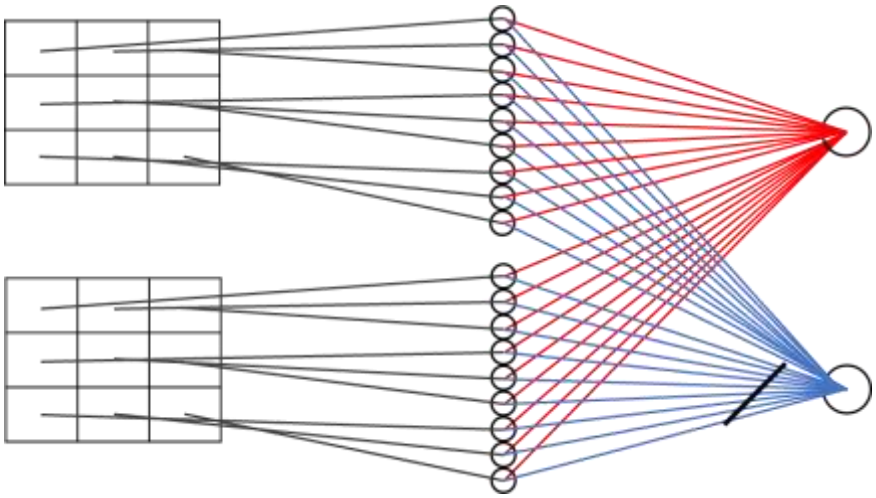


6. ábra
A konvolúciós és fully connected rétegek analóg működése indokolja, hogy CCN-ek esetében súlyok helyett kernelenként regularizáljunk.

Feltevésem szerint az általam korában definiált $Tenzor\ L_0$ regularizáció alkalmazása több szempontból is célravezetőbb, mint a szokásos L_1 és L_2^2. Először is így nem 0 súlyokat, hanem nullmátrix filterket kapunk, ami jelentősen redukálja a számítási erőforrásigényeket. Nullmátrix-szal való konvolúció eredménye ugyanis minden pixelében 0 kimenetet eredményez, tehát ezt ki sem kell számítani.

Az ilyen módon kinullázott $\boldsymbol{W}_{i,j}^{(l+1)}$ interpretációja az, hogy az $\boldsymbol{L}_j^l$ csatorna nem vesz részt az $\boldsymbol{L}_i^{(l+1)}$ csatorna értékeinek definiálásában. Ez azért lehet hatékonyabb, mint a [10]-ben javasol módszer, mert az azzal analóg megoldás az lenne, ha bizonyos megfontolások mentén csatornákat dobnánk el. Ezzel a módszerrel viszont nem dobjuk el, csak egymást követő rétegbeli csatornapárok közötti függőség szűnik meg. Ez pedig olyan pozitív következményekkel járhat, hogy csatornák diverzifikálódni tudnak a későbbi rétegek, illetve maga a hálónak tanított feladat igényei szerint.

Hasonló megfontolásból érdemes lehet a kilapítás utáni első réteg súlyait is csoportosítva regularizálni. Ugyanis a kilapítás lényegében csak egy implementációs trükk, arra, hogy átmenetet képezzen a jellemző kiemelést végző konvolúciós és a végső döntést hozó fully connected rétegek között. Igazából az utolsó réteg csatornáit látjuk sorosítva. Ezt az adathalmazt ugyanúgy értelmetlen pixelenként feldolgozni, sokkal hasznosabb lehet, ha a hiba-visszaterjesztés során azt a kérdést tesszük fel, hogy az adott csatorna fontos szerepet játszik-e adott neuron szempontjából. Érdemes azt is figyelembe venni, hogy a regularizációt sok esetben robosztusság növelése céljából alkalmazzuk. Viszont egy-egy szűrés szomszédos pixelei között a szűrések természetéből fakadóan jelentős korreláció adódik, ezért a pixelenkénti ritkító súlyregularizáció nem csökkentené a háló zavarérzékenységét, hanem épp ellenkezőleg, minden bizonnyal növelné.



7. ábra
Az első fully connected réteg esetében szintén érdemesebb logikailag egybetartozónak tekinteni egy csatorna pixeleit és ez alapján csoportosan regularizálni a súlyokat.

4.2.4 Megvalósítás

Kerasban lehetőség van callback függvények megadására. Ezek olyan függvények, amiket a felhasználó definiál és be tudja állítani, hogy a tanítás során mikor fussanak le. Tipikus példa a használatára a korai leállás implementálása, aminek a lényege, hogy ha előre definiált számú epoch óta nem javult a validáló mintákon a háló hibája, akkor leállítja a tanítást. Ezzel elkerülve a tanító adatokra történő túlilleszkedést és ezzel együtt a háló általánosító képességének a romlását.

Ilyen callback függvény segítségével valósítottam meg a $Tenzor\ L_0$ regularizációt is. Ez azért tehető meg, mivel a proximity gradient minden iterációjának első lépése pontosan a gradient descent által is végzett súly frissítés, tehát ezt a keretrendszer el is végzi. Ha pedig

a callbackben implementáljuk a proximity gradient algoritmus második lépését, akkor pontosan az elvárt működést kapjuk.

Mivel a callback-en belül minden szűrő súlymátrixát el lehet kérni, a konvolúciók esetében a következőképpen jártam el. Minden epoch végén minden tanult konvolúciós szűrőnek kiszámoltam a Frobenius-normáját és ha az egy, a tanítás elején megadott hipeparaméternél kisebb, akkor a mátrixot kinulláztam.

A kilapítás utáni első réteg regularizációja esetén ehhez hasonlóan kell eljárni (csak arra az implementációs részletre kell ügyelni, hogy helyesen állítsuk vissza a sorosított mátrix dimenzióit).

4.2.5 Tapasztalatok

A vizsgált háló struktúrája a következő:

0. réteg: STN réteg (kimeneti kép: $30x30$, bilineáris interpoláció)
1. réteg: 2D konvolúció (ablakméret: $3x3$, kimeneti csatornák száma: 32, aktiváció: relu)
2. réteg: maxpooling, (ablakméret: $2x2$)
3. réteg: 2D konvolúció (ablakméret: $3x3$, kimeneti csatornák száma: C_n, aktiváció: relu)
4. réteg: maxpooling, (ablakméret: $2x2$)
5. réteg: flatten
6. réteg: fully connected (neuronok száma: 10, aktiváció: softmax)

ahol a C_n értéket 32-re állítva már korábban vizsgált hálót kapunk. Természetesen minimális eltéréseket tapasztalhatunk az újramérések során. Minden esetben egyszerre két rétegen próbáltam ki a $Tenzor\ L_0$ regularizációt: az utolsó konvolúciós rétegen és a kilapítás utáni első rétegen. Tapasztalataim alapján a regularizáció hatásosabb, ha ezekhez két eltérő hiperparamétert használunk, így a konvolúciós réteg küszöbértékét H_c-vel a fullyconnected rétegét H_f-fel jelölöm. A mérési eredmények az *6. táblázat*ban találhatóak.

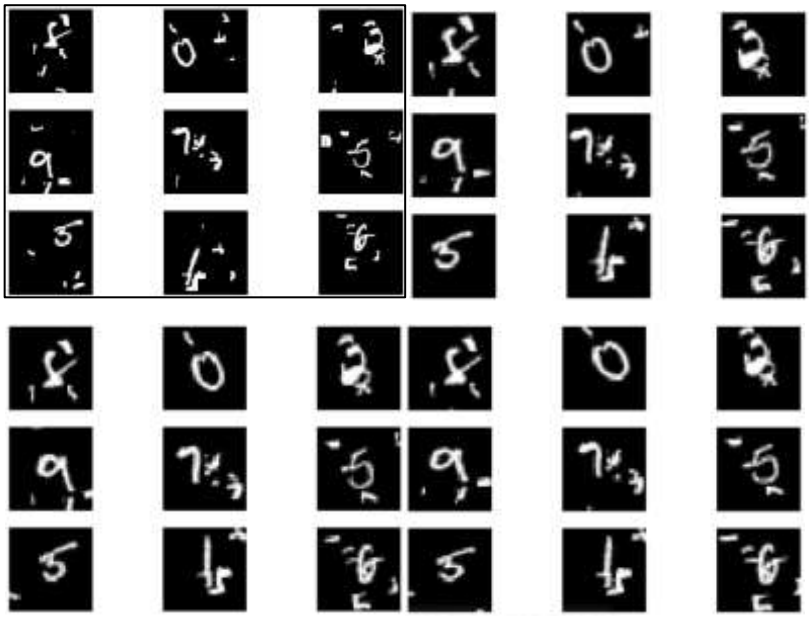
C_n	32	64	128
tanító	93,71%	94,49%	93,51%
validáló	92,19%	92,59%	92,03%
tesztelő	92,15%	92,37%	91,57%
H_c	0	0.205	0.215
H_f	0	0.285	0.28
kinullázott kernelek	0 / 1024	1078 / 2048	3126 / 4096
kinullázott csatornák	0 / 320	352 / 640	965 / 1280

6. táblázat
A $Tenzor\ L_0$ regularizáció hatása különböző paraméter értékek esetén.

A mérések során a hieprparamétereket úgy hangoltam, hogy a megmaradó, nem csupa 0 szűrők / teljesen összekötött rétegbeli kapcsolatok súly paramétereinek a száma minél közelebb essen a regulrizálatlan 32 csatornás háló súlyainak a számához. Tapasztalati eredmények alapján a pontosságban nincs szignifikáns eltérés. Ezért kijelenthető, hogy

a jelenlegi, nem optimális implementáció mellett is képes volt a háló úgy redukálni a paramétereinek a számát, hogy hasonló eredményt ért el, mint a végig ugyanannyi paraméterrel tanított háló. Várhatóan ez az optimalizációs eljárás módosításával még javulna is. Továbbá azt is érdemes szem előtt tartani, hogy a páronként több független csatorna miatt az egyes szűrések specifikusabb transzformációkat tanulnak meg, mely alapján a háló döntésének értelmezése is javulhat.

A három háló STN rétegének kimenetét a *8. ábrá*n láthatjuk. Ez alapján kvalitatíve a második kettő között nem mutatkozik különbség, de a 32 csatornás esetnél mindkettőn jobb pozícióba kerültek a számjegyek: mind az eltolásukat, mind az átskálázásukat tekintve jobb eredményt kaptunk.

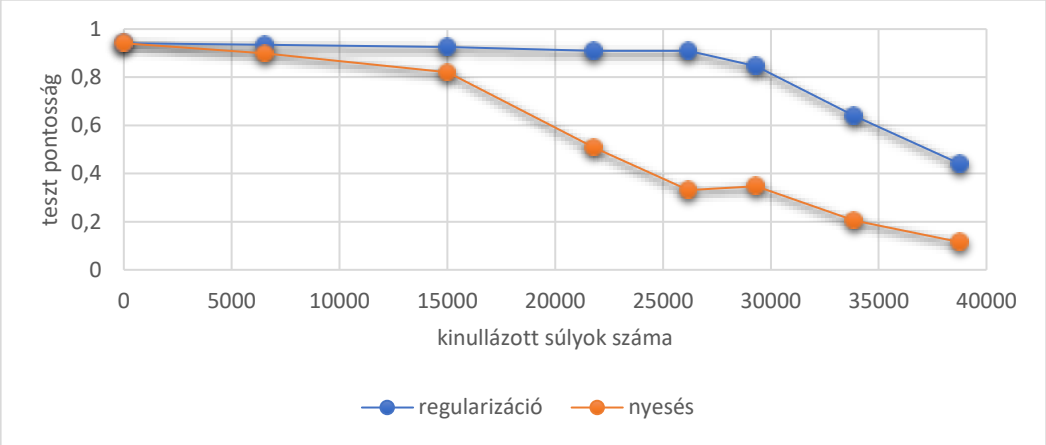


8. ábra:
A három háló STN rétegének kimenetei. A keretben 9 bemeneti kép látható. A többi az ezekhez a képekhez tartozó kimenetek a 32 (jobb felső), 64 (bal alsó) illetve 128 (jobb alsó) csatornás esetben.

Ami még megállapítható, hogy a két konvolúciós réteg kimenetén valóban sokkal diverzebb képek jelennek meg a regularizált több csatornás hálók esetén. Ez igaz az első konvolúciós rétegre is, melynek szűrőit nem regularizáltuk. Ennek magyarázata, hogy ezek a csatornák is specializálódni tudtak a háló azon későbbi részeihez, melyek feldolgozták őket. A diverzifikációra a függelékben láthatunk mintát.

Az ezt követő mérés során a $Tenzor\ L_0$ regularizáció hatékonyságát hasonlítottam össze egy a szakirodalmoban [9, 10] eddig javasolt módszerekhez hasonlóan működő eljárással. Ehhez az előző vizsgálat során használt hálón nem végeztem strukturális változtatást.

Az összehasonlítás alapját képző paramter nyesési módszer esetén a betanított háló súlymátrixait törlöm Frobénus normájuk szerinti növekvő sorrendben. Ezzel a módszerrel pontosan megadható, hogy hány súlymátrixot akarunk elhagyni, így könnyen összehasonlítható a $Tenzor\ L_0$ regularizáció eredményével.



9. ábra
Az x tengelyen a $Tensor\ L_0$ segítségével kinullázott paraméterek száma
(ez konvolúciós kernelenként $3 * 3$, fully connected csoportosított súlyainál $6 * 6$ súly egy mátrix kinullázásakor)
az y tengelyen pedig az így kapott teszt pontosság a ClutteredMNIST adatbázison.

A diagrammon jól látszik, hogy a nyeséshez képest körülbelül kétszeres mértékű paraméterredukció (30000 eldobott paraméter) esetén csökken a pontosság 80% alá. Az is leolvasható, hogy amikor a nyeséssel redukált háló már csak 10% körüli teljesítményre képes (vagyis véletlenszerű predikcióval azonos pontosságú) a $Tenzor\ L_0$-val tanított háló még mindig 40% felett teljesít. Továbbá, hogy a töréspont a regularitázió teljesítményében csak kevéssel az után következik, miután kinullázható paraméterek több mint felét eliminálta. Részletesebb eredmények a függelékben találhatók.

4.2.6 Összefoglalás

A javasolt regularizációs módszerrel tehát kvantitatíve jobbnak bizonyult a háló paraméterszám csökkentésében, mintha a szakirodalom által eddig tárgyalt tanítás utáni nyesét alkalmaznánk. Ezen felül bizonyos megfontolások mentén kvalitatíve is hatékonyabb megoldás születéséhez vezetett. Az így kapott hálóban a részfeladatok jobban különválaszthatók, így a predikciók magyarázata is könnyebbé válhat. Valószínűnek tartom, hogy más problémákra, illetve az IRWL2-es eljárással analóg optimalizáció alkalmazás esetén tovább javulna a regularizáció hatásossága.

5 Összefoglalás, továbbfejlesztési lehetőségek

A dolgozat során olyan eljárásokat vizsgáltam meg és analizáltam, melyek alkalmazásával javítható a konvolúciós neurális hálók döntéseinek interpretálhatósága, illetve csökkenthető a háló komplexitása a pontosság tolerálható mértékű redukciója mellett.

A tudás desztillálás széleskörűen elfogadott hálókompressziós módszer, melynek hatásosságát jelentősen sikerült meghaladni más megközelítések alkalmazásával. Ezek közül az első taglalt módszer a bemenetek uniformizálásához kapcsolódik, melyet automatizáltan az STN réteg alkalmazásával realizáltam.

Mivel az STN réteg használatával logikailag is sikerül a háló által végzett műveletsort két részre bontani, így ezt a réteget nem kell interpretálni. Az STN által végzett uniformizálás hatására réteg kimenetére illesztett háló komplexitása nagyságrendileg csökken. Valamint a regisztráció következtében a megtanult szűrők között csökkenő redundancia miatt az interpretálhatóság is egyszerűsödik. A módszer hatásosságát kvalitatívan is megvizsgáltam, mely alapján a paraméterek számának 100-ad részre történő redukciója mellett a háló pontossága hozzávetőleg 5%-al csökkent a Cluttered MNIST adathalmazon. Sikerült olyan hálót is konstruálnom, mely csak egy konvolúciós és egy teljesen összekötött réteget tartalmazott az STN-en kívül, és az adathalmazon 90% feletti pontosságot ért el.

A dolgozatban javaslatot tettem az STN-re épülő háló hatékonyságának további fokozására direkt regularizáció használatával. A szakirodalomban széleskörűen tárgyalt, ezért közismert elemenkénti regularizációs eljárások mellett a csoportonkénti regularizációs eljárások használatát is megvizsgáltam. Összehasonlítottam elvi megfontolások mentén a taglalt regularizációs módszereket, továbbá bevezettem egy ismereteim szerint más által eddig még nem vizsgált regularizációs büntetőfüggvényt, mely a $Tensor\ L_0$ normán alapult. Az implementált regularizációval nagy mértékben csökkentettem az STN alapú háló komplexitását. Ezen eljárás implementációs problémájával is foglalkoztam, mely analízis alapján a proximity gradient alapú numerikus optimalizációt választottam. Noha rámutattam hiányosságára, ezen optimalizáció alkalmazásával is jelentősen sikerült meghaladni a szakirodalomban publikált nyesési módszerek hatásosságát. A pontosság $3{,}21\%$-os esése mellett a szűrések 59%-ától sikerült megszabadultam. Lehetőségként javaslatot tettem az optimalizálási probléma újrasúlyozott Frobenius-normás regularizációra történő visszavezetésére (IRWL2-n alapuló optimalizáció), melyet a dolgozatban bemutatott kutatás folyományaként tervezek implementálni.

6 Irodalomjegyzék

- [1] Geoffrey Hinton, Oriol Vinyals, Jeff Dean „*Distilling the Knowledge in a Neural Network,*" arxiv 1503.02531, 2015
- [2] Nicholas Frosst, Geoffrey Hinton „*Distilling a Neural Network Into a Soft Decision Tree,*" CEX 2017
- [3] Ozan Irsoy, Olcay Taner Yıldız, Ethem Alpaydın, „ *Soft Decision Trees*", ICPR 2012
- [4] Max Jaderberg, Karen Simonyan, Andrew Zisserman, Koray Kavukcuoglu „*Spatial Transformer Network,*" NIPS 2015, vol. 28, pp. 2017-2025, 2015.
- [5] Alexey Kurakin, Ian Goodfellow, Samy Bengio „*Adversarial examples in the physical world,*" arXiv 1607.02533, 2016
- [6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun „*Deep Residual Learning for Image Recognition,*" CVPR 2016, pp. 770-778
- [7] Vincent Vanhoucke, Andrew Senior, Mark Z Mao, "Improving the speed of neural networks on CPUs," NIPS 2011, 2011.
- [8] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, Pritish Narayanan, "*Deep learning with limited numerical precision*," ICML 2015, pp. 1737–1746, 2015
- [9] Yann Le Cun, John S. Denker, Sara A. Solla, „Advances in neural information processing systems 2," D. S. Touretzky, Ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990, ch. Optimal Brain Damage, pp. 598–605.
- [10] Suraj Srinivas, R. Venkatesh Babu, „*Data-free Parameter Pruning for Deep Neural Networks,*" BMVC 2015, pp. 31.1–31.12, 2015
- [11] Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, Rob Fergus, „*Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation,*" NIPS 2014, pp. 1269–1277, 2014
- [12] Peter Sollich, Anders Krogh, „*Ensemble Methods in Machine Learning: How overfitting can be useful,*" NIPS 1996, pp. 190-196, The MIT Press, 1996.
- [13] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, „*ImageNet Classification with Deep Convolutional Neural Networks,*" NIPS 2012
- [14] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, Wojciech Samek, „*On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation,*" Doi 10.1371/journal.pone.0130140, 2015
- [15] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna, „*Rethinking the Inception Architecture for Computer Vision*", arXiv 1512.00567, 2015
- [16] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, Kurt Keutzer, „*SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size*", arXiv 1602.07360, 2016
- [17] Alfredo Canziani, Adam Paszke, Eugenio Culurciello, „*An Analysis of Deep Neural Network Models for Practical Applications*", arXiv 1605.07678, 2016

- [18] Wenling Shang, Kihyuk Sohn, Diogo Almeida, Honglak Lee, „*Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units*", arXiv 1603.05201, 2016
- [19] Max Pechyonkin, „Understanding Hinton's Capsule Networks. Part I: Intuition." medium.com/ai³-theory-practice-business/understanding-hintons-capsule-networks-part-i-intuition-b4b559d1159b
- [20] Hui Zou, „*The Adaptive Lasso and Its Oracle Properties*", Doi 10.1198/016214506000000735
- [21] Emmanuel J. Candes, Michael B. Wakin, Stephen P. Boyd, „*Enhancing Sparsity by Reweighted L1 Minimization*", arXiv 0711.1612, 2017
- [22] Sara Sabour, Nicholas Frosst, Geoffrey E Hinton, „*Dynamic Routing Between Capsules*", arXiv 1710.09829, 2017

7 Függelék

A regularizáció és a nyesés közti különbséget ábrázoló diagram mérési pontjaihoz tartozó mérési eredmények részletesen:

algoritmus: r=regularizáció, ny=nyesés

tanító, validáló, tesztelő: százalékban értendő, az adott adathalmazon vett pontosság

kernelek: a regularizált konvolúciós réteg kinullázott kerneleinek a száma az eredeti 2080* közül

csatornák: a regularizált fully connected réteg súlymátrixai közül a kinullázottak száma (egy súlymátrix egy kilapítás előtti csatorna pixeleinek a súlyát tartalmazza)

algoritmus	r	ny	r	ny	r	ny	r	ny	r	ny	r	ny	r	ny	r	ny
tanító	96	96	95	91	94	83	93	51	92	34	86	35	65	21	45	11
validáló	94	94	94	90	93	81	91	50	91	33	85	35	63	20	44	10
tesztelő	**94**, **12**	94	93	90	92	82	91	50	**90**, **91**	33	85	35	64	21	44	11
kernelek	0		453		841		1134		1225		1263		1479		1837	
csatornák	0		69		207		322		421		499		571		618	

*az háló összes szűrőjének száma a kiinduláskor: 32+32*64 = 2080

Diverzifikáció:

a b
c d
e f

a: második konvolúciós réteg 32 csatorna, első konvolúciós réteg kimeneti csatornái
b: második konvolúciós réteg 32 csatorna, második konvolúciós réteg kimeneti csatornái
c: második konvolúciós réteg 64 csatorna, első konvolúciós réteg kimeneti csatornái
d: második konvolúciós réteg 64 csatorna, második konvolúciós réteg kimeneti csatornái
e: második konvolúciós réteg 128 csatorna, első konvolúciós réteg kimeneti csatornái
f: második konvolúciós réteg 128 csatorna, második konvolúciós réteg kimeneti csatornái