

# Általános információk, a diplomaterv szerkezete

A diplomaterv szerkezete a BME Villamosmérnöki és Informatikai Karán:

1. Diplomaterv feladatkiírás
2. Címoldal
3. Tartalomjegyzék
4. A diplomatervező nyilatkozata az önálló munkáról és az elektronikus adatok kezeléséről
5. Tartalmi összefoglaló magyarul és angolul
6. Bevezetés: a feladat értelmezése, a tervezés célja, a feladat indokoltsága, a diplomaterv felépítésének rövid összefoglalása
7. A feladatkiírás pontosítása és részletes elemzése
8. Előzmények (irodalomkutatás, hasonló alkotások), az ezekből levonható következtetések
9. A tervezés részletes leírása, a döntési lehetőségek értékelése és a választott megoldások indoklása
10. A megtervezett műszaki alkotás értékelése, kritikai elemzése, továbbfejlesztési lehetőségek
11. Esetleges köszönetnyilvánítások
12. Részletes és pontos irodalomjegyzék
13. Függelék(ek)

Felhasználható a következő oldaltól kezdődő L<sup>A</sup>T<sub>E</sub>X-Diplomaterv sablon dokumentum tartalma.

A diplomaterv szabványos méretű A4-es lapokra kerüljön. Az oldalak tükörmargóval készüljenek ( mindenhol 2.5cm, baloldalon 1cm-es kötéssel). Az alapértelmezett betűkészlet a 12 pontos Times New Roman, másfeles sorközzel.

Minden oldalon - az első négy szerkezeti elem kivételével - szerepelnie kell az oldalszámnak.

A fejezeteket decimális beosztással kell ellát ni. Az ábrákat a megfelelő helyre be kell illeszteni, fejezetenként decimális számmal és kifejező címmel kell ellát ni. A fejezeteket decimális alaosztással számozzuk, maximálisan 3 alaosztás mélységen (pl. 2.3.4.1.). Az ábrákat, táblázatokat és képleteket célszerű fejezetenként külön számozni (pl. 2.4. ábra, 4.2 táblázat vagy képlet nél (3.2)). A fejezetcímeket igazitsuk balra, a normál szövegnél viszont használunk sorkiegynítést. Az ábrákat, táblázatokat és a hozzájuk tartozó címet igazitsuk középre. A cím a jelölt rész alatt helyezkedjen el.

A képeket lehetőleg rajzoló programmal készítsék el, az egyenleteket egyenlet-szerkesztő segítségével írják le (A L<sup>A</sup>T<sub>E</sub>X ehhez kézenfekvő megoldásokat nyújt).

Az irodalomjegyzék szövegközi hivatkozása történhet a Harvard-rendszerben (a szerző és az évszám megadásával) vagy sorszámozva. A teljes lista névsor szerinti sorrendben a szöveg végén szerepeljen (sorszámoszott irodalmi hivatkozások esetén hivatkozási sorrendben). A szakirodalmi források címeit azonban minden eredeti nyelven kell megadni, esetleg zárójelben a fordítással. A listában szereplő valamennyi publikációra hivatkozni kell a szövegben (a L<sup>A</sup>T<sub>E</sub>X-sablon a Bib<sup>T</sup>EX segítségével mindezt automatikusan kezeli). minden publikáció a szerzők után a következő adatok szerepelnek: folyóirat cikkeknél a pontos cím, a folyóirat címe, évfolyam, szám, oldalszám tól-ig. A folyóirat címeket csak akkor rövidítsük, ha azok nagyon közismertek vagy nagyon hosszúak. Internet hivatkozások megadásakor fontos, hogy az elérési út előtt megadjuk az oldal tulajdonosát és tartalmát (mivel a link egy idő után akár elérhetetlenne is válhat), valamint az elérés időpontját.

Fontos:

- A szakdolgozat készítő / diplomatervező nyilatkozata (a jelen sablonban szereplő szövegtartalommal) kötelező előírás Karunkon ennek hiányában a szakdolgozat/diplomaterv nem bírálható és nem védhető !
- Mind a dolgozat, mind a melléklet maximálisan 15 MB méretű lehet !

Jó munkát, sikeres szakdolgozat készítést ill. diplomatervezést kívánunk !

## FELADATKIÍRÁS

A feladatkiírást a tanszéki adminisztrációban lehet átvenni, és a leadott munkába eredeti, tanszéki pecséttel ellátott és a tanszékvezető által aláírt lapot kell belefűzni (ezen oldal helyett, ez az oldal csak útmutatás). Az elektronikusan feltöltött dolgozatban már nem kell beleszerkeszteni ezt a feladatkiírást.



Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Méréstechnika és Információs Rendszerek Tanszék

# Mesterséges neurális hálózatok fejlesztése TensorFlow alapon

SZAKDOLGOZAT

*Készítette*  
Kemény Károly

*Konzulens*  
dr. Strausz György

2016. október 15.

# Tartalomjegyzék

<b>Kivonat</b>	<b>4</b>
<b>Abstract</b>	<b>5</b>
<b>Bevezető</b>	<b>6</b>
0.1. Motiváció . . . . .	6
0.2. A gépi tanulás igen rövid történelme . . . . .	7
<b>1. A Tensorflow[22] ökoszisztéma áttekintése</b>	<b>9</b>
1.1. A Tensorflow könyvtár. . . . .	9
1.2. A Tensorflow modellek monitorozása és hibamentesítése . . . . .	10
1.3. A Tensorflow futás közben . . . . .	10
<b>2. A Neurális hálózatokkal való képfeldolgozás lehetőségeinek áttekintése</b>	<b>12</b>
2.1. Az algoritmusok kiértékelése . . . . .	12
2.1.1. Az adathalmazok . . . . .	12
2.2. A Legelterjedtebb neurális hálózatok képfeldolgozáshoz . . . . .	14
2.2.1. A többrétegű perceptron (MLP) . . . . .	14
2.2.2. A Korlátozott Boltzmann Gép . . . . .	16
2.2.3. State of the art MLP hálózatok . . . . .	20
2.2.4. Konvoluciós hálózatok . . . . .	21
2.3. További érdekes irányok a neurális képfeldolgozásban . . . . .	24
<b>3. Hálózatok implementálása és elemzése tensorflowban</b>	<b>27</b>
3.1. A baseline osztályozók . . . . .	27
3.2. A saját magam által kialakított fejlesztőkörnyezet . . . . .	27
3.3. A kísérletek alatt használt optimizátorok . . . . .	28
3.4. A saját implementációk bemutatása . . . . .	28
3.4.1. A hálózatok monitorozása . . . . .	28
3.4.2. A logiszikus regresszió . . . . .	29
3.4.3. A többrétegű perceptron . . . . .	29
3.4.4. Az RBM hálózat . . . . .	30
3.4.5. A DBN struktúra . . . . .	31
3.4.6. Hibrid modellek . . . . .	32

3.4.7. A Konvoluciós modell . . . . .	33
3.4.8. A konvoluciós modell skálázása . . . . .	34
<b>4. A mérési eredményeim összegzése</b>	<b>37</b>
4.1. A keretrendszer általános teljesítménye . . . . .	38
4.2. Rendszer szintű mérések . . . . .	38
4.3. A baseline eredmények ismertetése . . . . .	40
4.4. Az RBM struktúra optimalizálása . . . . .	40
4.5. Az MLP-vel elvégzett méréseim eredményei . . . . .	41
4.5.1. Az MNIST mérések . . . . .	41
4.5.2. A CIFAR-10 mérések . . . . .	43
4.5.3. Az előtanítás eredményeinek összefoglalása . . . . .	43
4.6. A konvoluciós hálózatokkal elvégzett méréseim eredményei . . . . .	45
4.7. A konvoluciós és az MLP hálózatok erőforrás igényének az összehasonlítása	46
4.8. A hibrid architektúrák eredményei . . . . .	47
4.9. A CNN és az SVM+RBM összehasonlítása . . . . .	50
4.10. A három megközelítés végső összehasonlítása . . . . .	50
<b>5. Az eredmények összefoglaló értékelése</b>	<b>51</b>
<b>Köszönetnyilvánítás</b>	<b>53</b>
<b>Irodalomjegyzék</b>	<b>55</b>

## HALLGATÓI NYILATKOZAT

Alulírott *Kemény Károly*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot/ diplomatervet ([nem kívánt törlendő](#)) meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltetem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2016. október 15.

---

*Kemény Károly*  
hallgató

# Kivonat

A szakdolgozatomnak két súlypontja volt, az egyik hogy elmélyítsem a tudásomat a mesterséges neurális architektúrák területén elméleti síkon, a másik hogy elsajátítsam a Tensorflow - ami egy elosztott, gráf alapú numerikus könyvtár a Google-től - ökoszisztémájának kezelését az előbb említett hálózatok programozásán keresztül. Mivel ez mindenféle keret nélkül hatalmas témakör lenne, ezért a dolgozat fókuszául a képosztályozást választottam.

A szakdolgozat először is arra a kérdésre próbál meg röviden választ adni hogy miért kell még ma is effektív tudással rendelkeznie az embernek a hálók működéséről ahhoz hogy valóban komplex rendszerekben használni tudja őket, még akkor is ha az eddig elkészült architektúrák nagy része sokszor előre elkészítve, sőt akár bizonyos feladatokra előre tanítva is elérhető. Ezután egy rövid történelmi összefoglalóval teszem kontextusba a dolgozat témakörét.

A dolgozat irodalom kutatással foglalkozó részében a tensorflow ökoszisztémáját, és a mesterséges neurális hálózatokkal való képosztályozás elméletét tekintem át, véleményem szerint az összes fő témakört érintve. Ezek sorrendben az előtanulás nélküli többrétegű perceptronok, az előtanított többrétegű perceptronok, a konvoluciós hálózatok, és a hibrid architektúrák. A hibrid architektúrákon belül kitérek két megközelítésre is. Az első amikor egy generatív architektúrát kombinálunk egy teljesen külön álló diszkriminatív osztályozával, esetben egy korlátozott boltzmann gépet egy szupport vektor géppel. A második amikor maga a neurális architektúra hibrid, tehát egyben modellben ötvözi a generatív és a diszkriminatív megközelítést. Ez a hibrid korlátozott boltzmann gépek megközelítése ami egy igen új fejlemény. Ezek után ismertetem a terület egy-két számomra izgalmas és új fejleményét.

A kísérleti részben az előbbiekben leírt elméleti részből kísérletezem számos architektúrával az MNIST és a CIFAR-10 adathalmazokon. Itt inkább előtérbe helyezem az egyes architektúrákat, és a hálózatok futás idejű karakterszitikáit, és elhanyagolom az olyan témaköröket mint a bemeneti adatok előfeldolgozása, a tanítás korai megállítása vagy a hiperparaméter térben való effektív keresés és ezek kombinációinak a (kereszt)-validációja. Az előbb említett pontok természetesen egy valós ipari környezetben nagy jelentőséggel bírnának, de egyenként is érdemesek lennének akár egy egész TDK, szakdolgozat vagy diploma témának. De sajnos elvennék a fókuszt attól amivel foglalkozni szerettem volna, nevezetesen a különböző modellekkel való kísérletezéstől.

A dolgozatot az előbbi fejezetekből levont tanulságokkal zárom.

# Abstract

I emphasised two main aspects in my B.Sc thesis. On one hand, I have wanted to deepen my theoretical knowledge in the area of artificial neural networks. On the other, I have wanted to master Tensorflow - a computational graph based distributed numerical framework from Google - through the programming of neural networks. However, it would have been an enormous topic to cover without any constraints, so I have chosen image classification as my topic of focus.

First, I explain the motivation behind my work. The reason why I think that it is still relevant to have profound knowledge of neural architectures if someone wants to use them in a complex application, even though there are a lot of architectures out there ready-made or even pretrained for a specific task.

In the literature research section I have glanced through the ecosystem of Tensorflow and the theory behind neural image classification, covering the most important topics I believe. These were as follows: multilayer perceptrons with and without pretraining, convolutional networks and the hybrid architectures. I have written briefly over two approaches regarding the hybrid solutions. The first one is when one has separately a generative model and a discriminative classifier, in my case this would be a restricted boltzmann machine for the former and a support vector machine for the latter. The second one is when these two are fusioned into one model, which is fascinating to me. I have found two such approaches, the hybrid restricted boltzmann machine (Larochelle 2008), and its stacked version the stacked boltzmann expert network (Alexander G. 2015). In the last section of my research I have highlighted some newer, intriguing advancements in the field.

In the measurements section of my thesis I did plenty of experimentation on some of the models that I have mentioned before, for evaluation I have used the MNIST and the CIFAR-10 datasets. I had to make a decisive decision which parts of a full experimental setting to neglect. I have chosen to pay little to no attention to the preprocessing of data, optimal early stopping algorithms for training, effective search and evaluation in the hyperparameter space. These could make good candidates for a thesis on their own, but they would have gravely drawn the attention from my main goal which was to experiment on a wide range of models.

I conclude my thesis with a brief summarization of what I have learned from the previous sections.

# Bevezető

## 0.1. Motiváció

**A neurális hálózat** A neurális hálózat egy számítási modell amelyet számos, algoritmikailag nehéz problémára sikeresen lehet alkalmazni. A fő alkalmazási területeik: osztályozási feladatok, regressziós feladatok, dimenzió csökkentés (kernel PCA), jellemző kiemelés. Ezt a modellt sikeresen alkalmazták már az ipar számos területén, a teljesség igénye nélkül:

1. *képfelismerés*: Ezen a területen talán a legsokrétűbb a felhasználásuk az egyszerű OCR rendszerektől kezdve egészen a rákos daganatok detektálásáig elterjedtek.
2. *idősor előrejelzés*: Komplex idősor előrejelzésnél is előszeretettel használják óket, amikor a sok változót és azoknak összefüggését bonyolultságuknál fogva már nem lehet klasszikus statisztikai módszerekkel megragadni.
3. *beszédszintetizálás*: A 2016 szeptemberében publikált WaveNet struktúra 50%-ot volt képes javítani Mean Opinion Scoreban az eddigi state-of-the-art beszédszintetizáló rendszereken. Ezt a javulást nyelvfüggetlenül, angolban és mandarin kínaiiban is képes volt tartani. [25]
4. *szabályozó algoritmusok*: Szintén 2016-ban a DeepMind AI alkalmazása a Google egy adatközpontjának a hűtés vezérlésében 40%-os esést eredményezett az erre fordított kiadásokban.

Ezek véleményem szerint mind kifejezetten izgalmas eredmények, tisztán látszik hogy a terület él, fejlődik és napról napra formálja a jelfeldolgozásról alkotott képünket.

**Jelen Dolgozat célja** A dolgozat fő csapásiranya egy áttekintő kép alkotása a neurális hálózatok felhasználásáról képosztályozási problémák esetében. A dolgozatomban arra törekszem hogy a neurális hálózatok gyakorlati alkalmazását ezen az területen keresztül ismerjem meg. Mivel tanulmányaim során ilyesféle - matematikai vonatkozású - programozással nem találkoztam jelentős mértékben, ezért ez egy jó alkalom arra hogy lássam hogy komplex matematikai absztrakciók hogyan öltének testet szoftverként. Például a sztochasztikus generativ vagy a konvoluciós neurális hálózatok hogyan fordíthatóak le effektív programmá, és hogyan lehet belőlük értékes alkalmazásokat készíteni. A szakdolgozat egy hosszabb projekt része, ahol egy Tegra mobil chipen kell majd képosztályozó algoritmusokat alkalmazni ami egy robotrepülőgépen fog valós idejű döntéseket hozni. A dolgozat írása alatt értékelődik

ki hogy a projektben a Tensorflow reális alternatíva lesz-e a további munkához, illetve hogy melyik képosztályzó eljárás felelne meg legjobban a céljainknak.

**Érdeklődésem a neurális hálózatok iránt** Az önálló labor munkám során és a szakirányomon oktatott kooperatív és tanuló rendszerek tárgyban találkoztam először ezzel a megközelítéssel. Az egyre modernebb ajánló algoritmusok keresése közepedte rá kellett eszmélnem hogy az ajánlórendszer jövője is összefonódik a neurális hálózatokéval, amíg a 2015ös RecSys konferencián egyetlen papir sem szólt a mély tanulásról, az idei konferencia publikációinak a negyede a mély modellek felhasználását tagalta. Igy ez a dolgozat habár első látásra nem is látszik, de az önálló laboratóriumi projekt munkám továbbvitele. Egy potenciálisan érdekes, még alig kutatott terület, hogy a termékek hibás és hiányos leírását hogyan lehet augmentálni a termékek feltöltött képeiből kinyert címke felhővel. Erre nagy valószínűséggel alkalmazható hálózatok lennének a regionális konvoluciós hálózatok amiket érintőlegesen tárgyalok majd a konvoluciós hálózatokat taglaló fejezetben.

**A feladat indokoltsága** Az olvasó jogosan teheti fel magában a kérdést hogy ha ezek a hálózatok már léteznek, sőt sok esetben konfiguráció nélkül "out of the box" jelleggel használhatóak, akkor mi ad létjogosultságot egy ilyen bevezető jellegű szakdolgozatnak? Nos, habár az előző pont igaz, mégis manapság minnél inkább érdemes tisztában lenni ezeknek az eszközöknek a képességeivel, és mind fontosabb a korlátaiival. Ha valaki egy saját alkalmazásban szeretné őket használni, akkor érdemes tudni hogy:

1. Az adott alkalmazáshoz milyen háló típusok használhatóak.
2. Van-e esetleg már előre tanított modell a feladatunkhoz.
3. Ha nincs akkor mennyi idő lenne betanítani egyet.
4. Mennyi helyet és számítást igényelnek az egyes modellek. (Ez erősen például függ a modell paraméter terének nagyságától)

A szakdolgozat végére reményeim szerint az egyetemi tárgyak anyagát továbbfejlesztve egy nagyobb rálátással fogok rendelkezni erre az izgalmas területre.

**A dolgozat kontextusa** Ahhoz hogy kontextusba helyezem jelen munkámat a bevezető további részében szeretnék egy rövid áttekintést adni a gépi tanulás történelméről.

## 0.2. A gépi tanulás igen rövid történelme

1. Az első elismerten tanuló gépet Arthur Samuelnek tulajdonítják 1955-ben, ez a konstrukció képes volt megtanulni dámajátékot játszani. Samuel algoritmusai heurisztikus keresési memóriát alkalmaztak annak érdekében hogy a saját tapasztalataikból tanuljanak. A hetvenes évek közepére ez a rendszer már képes volt emberi játékosok legyőzésére is.

2. Következő fontos pontként Frank Rosenblatt Perceptronját emelném ki, ez volt az első neurális hálózat, 1958-ban alkotta meg Rosenblatt az amerikai haditengerészet "US office of Naval Research" laboratóriumában. Már ezt is vizuális minták felismerésére alkották meg eredetileg.
3. A hetvenes éveket csak úgy emlegetik hogy a mesterséges intelligencia tele, miután Marvin Minsky 1969-ben rámutatott a Perceptron korlátaira az emberek elvesztették az érdeklődésüket a terület iránt. 1985-ben egy forradalmi újítás, a hibavisszaterjesztéses algoritmus (backpropagation algorithm [23]) törte meg a csendet és keltette fel az emberek érdeklődését újfent ezen számítási struktúrák iránt.
4. A kilencvenes években a neurális hálózatok újra kikerültek a középpontból, mert a statisztikusok által alkotott szupport vektor gépek (továbbiakban SVM) lényegesen jobb teljesíményt tudtak elérni, kevesebb tanítással mint a kor hálózatai.
5. A neurális hálózatok következő virágkorát napjainkban éljük, ennek egyik fő tényező a fejlett neurális struktúrák felfedezése, illetve az hogy a grafikus egységek és a számítási fürtök fejlődésének köszönhetően eddig elképzelhetetlen számítási kapacitás áll rendelkezésünkre a hálózataink tanítására. Ezzel szemben a kernel gépeken alapuló modellek nem tudtak a megnövekedett teljesítményt kihasználva a neurális hálózatokhoz hasonló pontosság növekedést elérni. A manapság a neurális hálózatok jelen vannak az élet minden területén ahol szükségünk van mintázatok intelligens felismerésére, lehet az a szolgáltatás hang, kép vagy akár szöveges dokumentumok. Google translate, Shazam, a netflix díj nyertes ajánlási algoritmus, csak hogy pár példát szemelvényezzük a számtalan közül.

**A dolgozat felépítése** A dolgozatomat az alábbi építem fel:

1. A tensorflow mint neurális rendszerek kutatására, fejlesztésére és éles üzembe helyezésére alkalmas platform bemutatása.
2. A tárgyterület irodalmának áttekintése, szemlézve a következő struktúrákat:
  - (a) Egyszerű többrétegű perceptron.
  - (b) Korlátozott boltzmann gépek, és variánsai
  - (c) Konvoluciós Neurális hálózatok
  - (d) A friss fejlemények a neurális képosztályozás területén.
3. A saját fejlesztésem bemutatása, amely egy két egyszerűbb struktúra implementálása és vizsgálata a fent bemutatottak közül, a tensorflow könyvtárral.
4. A mérési eredményeim kiértékelése, tanulságok levonása.

## 1. fejezet

# A Tensorflow[22] ökoszisztéma áttekintése

### 1.1. A Tensorflow könyvtár.

**A technológiai választás indoklása** A modern szoftvermérnöki munka szerves része a rendelkezésre álló eszközök garmadájából a legmegfelelőbb kiválasztása. Ez a kínálat mérétéke, az információk elszórtsága és ellentmondásossága miatt koránt sem egy egyszerű feladat. A címből talán úgy sejlik hogy a tensorflow a munkámhoz már egy előre eldöntött választás volt, de ez a megállapítás koránt sem lenne helytálló. A szakdolgozatom nulladik lépseként számos más - a neurális hálózatok fejlesztését támogató - könyvtárat vettem szemügyre. A teljesség igénye nélkül: Caffee, Chainer, CNTK, Matlab, Tensorflow, Thenao, Torch. A következőkben szeretném bemutatni az általam választott eszköz a Tensorflow felépítését, és ezzel mintegy implicit módon megindokolni hogy szerintem miért ez a megfelelő eszköz a neurális hálózatokkal való munkához.

**Bevezető** Munkám során a Tensorflow nevű könyvtárral dolgoztam. A Tensorflow egy elosztott, számítási gráf alapú numerikus könyvtár. A Goolge Deep Mind kutatócsoport szakemberei hozták létre azzal a céllal hogy saját modelljeiket fejlesszék és értékeljék ki benne. A könyvtár a DistBelief nevű keretrendszer egyenesági leszámazottjának tekinthető. A DistBelief csendben meghúzódva, de ott dolgozik a fejlett világ majdnem minden emberének az élete mögött, lévén hogy az Alphabet cégcsoport (A Google feldarabolásának utódvállalait összefogó holding) több mint 50 csapata adaptálta, és vértezte fel általa intelligenciával alkalmazását. Pár ismertebbet kiemelve: Google Search, Adwords, Google Maps, StreetView, Youtube, és természetesen a Google Translate. Miután évek tapasztalata gyülemlet fel az első generációs könyvtárak használata során, úgy érezték itt az idő hogy - szakítva a technológiai teherrel amit az első generáció hibái miatt magukkal hordoztak - létrehozzák a következő generációs gépi tanulás rendszerüket, ez lett a Tensorflow aminek a fő célja skálázható, elosztott gépi tanulási algoritmusok (főképp neurális hálózatok) fejlesztése.

**A Tensorflow alapgondolata.** Mint már említettem a Tensorflow könyvtárban az ember a modelljeit egy adatfolyam-szerű számítási gráfként definiálhatja. Ennek a megközelítésnek az a nagy előnye hogy nagyon jó skálázódási tulajdonságokkal rendelkezik. Miután a gráf csomópontjai az egyes számítások, ezek adott esetben hatékony módon szétoszthatók különböző eszközök, vagy akár egész gépek között szerverparkokban. A könyvtár ezen tulajdonságának még egy hosszabb részt fogunk később szentelni. A létrejött gráfra mint egy alaprajzra érdemes gondolni, amit aztán az egyes munkamenetek (session-ök) példányosítanak. Ezek a munkamenetek inicializálják a változókat, és a munkamenet képes a gráf egyes csomópontjait lefuttatni, amik igény vezérelt módon minden a bemenetükre kapcsolódó csúcsot lefuttatnak amíg el nem érnek egy bemenetig vagy egy kívülről betáplált változóig. Miután a csomópont sikeresen lefutott a kimenetét a csatolt nyelv egy változójaként adja vissza, például egy Python vagy C++ tömbként. Fontos megjegyezni hogy futás közben a gráffal nem lehet érintkezni, az egy atomi egységként fut le, hogy minnél jobban ki lehessen optimalizálni a számításokat.

## 1.2. A Tensorflow modellek monitorozása és hibamentesítése

**A Tensorboard** Mivel a modern gépi tanulás modellek hihetetlen összetettséggel bírnak, és nagyon sok mozgó alkatrészük van, ezért természetesen felmerül az igény hogy egy ilyen újszerű keretrendszerben ipari erősségű monitorozó és hibakereső funkciók kerüljenek bele. Ezeket az elvárásokat a Tensorflow esetében a mellékelt Tensorboard alrendszer teljesíti, amelyet munkám során én is extenzíven használtam. Ebből kifolyólag most nem is bocsájtanám bővebb tárgyalásra, hanem majd az önálló munka szekcióban ismertetnémm.

## 1.3. A Tensorflow futás közben

**A Tensorflow serving kiszolgáló rendszer** Miután a kutatólaborokból kikerülték az új gépi tanulás modellek nem elég őket csak publikálni, a cégek komoly hasznót remélnek tőlük. A Google is kijelentette hogy "Information Retrieval first company" helyett ők most már egy "AI first company". Ez természetesen egy hozzá illő infrastruktúra nélkül elképzelhetetlen. Ezért is hozták létre a Tensorflowhoz a Tensorflow Servinget, ami egy flexibilis, magas rendelkezésre állású kiszolgálórendszer. A rendszer lehetővé teszi új architektúrák kipróbálását, üzembe helyezését és A/B tesztelését, miközben egy stabil, verziózott API-t biztosít a kliensek számára. Természetesen ez mit sem érne ha nem skálázna gond nélkül hatalmas magasságokba, ezért egyszerűen integrálható a Kubernetes névre hallgató docker alapú cluster kezelő rendszerrel.

**A tensorflow skálázodása** Érdemes belegondolni hogy az adatközpont TCP (DCTCP) vagy az Infiniband kapcsolatok adott esetben akár több gigabyteos sebességet érhetnek el, ugyanakkor a mátrix szorzás - ami a gépi tanuló algoritmusoknak egy kardinális eleme - négyzetes ordót igényel. Tehát sokkal jobban megéri részgráfokat a csomópontok között szétosztani és inkább a hálózati többlettel kalkulálni, mint hogy egyetlen gépre bízzuk a feladatokat. A másik végletben viszont egy másik elvárás helyezkedik el. Miután mondjuk

egy osztályozási feladatnál a modellünket megtanítottuk felismerni valamit, tegyük fel hogy például egy, a látássérült embereknek készített alkalmazásban felismerni az forgalomjelző lámpákat és azok állapotát, természetes lenne az igény hogy ezt a rászoruló magával tudja vinni. A háló ugyan az, a súlyokat már megtanultuk, de most az egész modellünket egy mobil eszközön kell futtatni. Ez az eszköz az inferenciát jácint könnyedséggel bírná, csak a tanulást nem tudtuk volna kivitelezni rajta. Mérnökként logikus az igény hogy ehhez ne kelljen mégegyszer lefejleszteni a modell-t, így időt és pénzt spórolva. Erre lehetőség van a keretrendszerrel.

**Mobil környezet** A Tensorflow támogatja a mobil eszközön való futást, az előbbi szcenárió a könyvtárral egy teljesen járható úttá válik. A modell-t asztali számítógépen fejlesztem, adatparkon tanítom, és egy mobil eszközön futtam, mindezt jelentősebb kód újraírás nélkül. Jó példája ennek a felhasználási módnak a Google Translate, legújabb, nagy felhajtást elérő verziója. Ez az alkalmazás a nyelvi modelleket a Google irdatlan infrastruktúráján tanul folyamatosan, miközben az inferenciát a telefonkészüléken futtatják lokálisan. Itt igazándiból két neurális háló is szerepet játszik, az egyik a szöveget ismeri fel a képen (feltehetően egy faster R-CNN), a másik pedig az effektív fordítást végzi.

## 2. fejezet

# A Neurális hálózatokkal való képfeldolgozás lehetőségeinek áttekintése

A kutatómunka egy jelentős részét tette ki a dolgozatomnak, mivel az évek folyamán nagyon sok sikeres és sikertelen kísérlet született annak érdekében hogy hogyan lehetne neurális hálózatokkal képi adatokat feldolgozni. Talán mondhatjuk hogy ezek a hálózatok a legsikeresebb képosztályozó eljárások, de korántsem triviális hogy mik az előnyeik, hátrányaik és az egyes típusok minden komplexitású jelekkel képesek megbírközni. Először szeretném bemutatni az adathalmazokat amiken ezeket az algoritmusokat kiértékelik, majd eljutni az többrétegű perceptronuktól a konvoluciós hálózatokig, végül a legújabb trendek ismertetésével zárni a fejezetet.

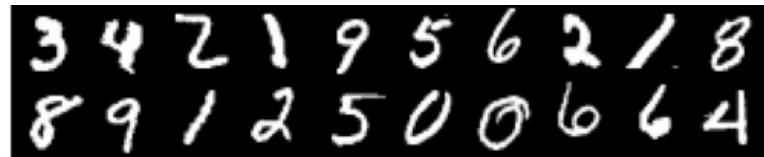
### 2.1. Az algoritmusok kiértékelése

#### 2.1.1. Az adathalmazok

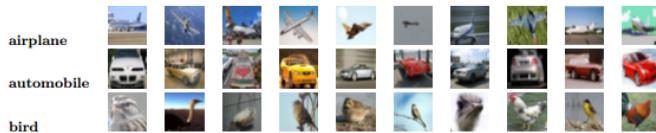
**Bevezetés** Mint a legtöbb kutatási területnek, ennek is vannak jól ismert "benchmark" adathalmazai, amelyek viszonyítási alapként lehetőséget biztosítanak az egymástól eltérő algoritmusok egymáshoz képesti kiértékelésére. Mivel ezekre az adathalmazokra sokat fogok hivatkozni, ezért szeretném őket egy-egy bekezdésben bemutatni.

**MNIST[2]** Az MNIST adatbázis fekete-fehér, 28x28 pixelre normalizált írott számjegyeket tartalmaz nullától kilencig, azaz tíz osztállyal rendelkezik. Az adatbázis 60'000 annotált tanító képet és 10'000 annotált teszt képet tartalmaz. Ez a legalapabb adathalmaz. Erre egy példát mutat az 2.1. ábra.

**CIFAR-10[1]** A CIFAR-10 egy jóval összetettebb adathalmaz, 60'0000 annotált 32x32 pixeles, színes képet tartalmaz. A képek 10 osztállyra vannak felosztva, osztályonként 6000 képpel. Az adathalmazból 50'000 kép van tanításra, és 10'000 tesztelésre fenntartva. Erre egy példát mutat a 2.2. ábra.



**2.1. ábra.** Példa az MNIST adathalmaz képeire. Forrás: <http://knowm.org/wp-content/uploads/Screen-Shot-2015-08-14-at-2.44.57-PM.png>.



**2.2. ábra.** Példa a CIFAR-10/100 adathalmaz képeire. Forrás: <http://www.cs.toronto.edu/~kriz/cifar.html>

**CIFAR-100[1]** A CIFAR-100 felépítése megegyezik a CIFAR-10el, de osztályrendszere az előbbinél lényegesen összetettebb, 100 osztályt tartalmaz és minden osztályhoz 600 kép tartozik, ezen felül még 20 általánosabb osztályba is be vannak sorolva a képek, hogy a hálózat általánosításáról következtetések lehessen levonni. Például az halak szuperosztályhoz tartozik a rája, lazac, stb. Példának úgyanúgy a CIFAR-10 mintája, a 2.2. ábra tekinthető.

**IMAGENET** Az IMAGENET a világ legnagyobb képgyűjteménye, a WordNet lexikális adatbázis szinoníma halmazai szerint vannak annotálva a képek. Jelenlegi statiszkái:

- 14'197'112 annotált kép
- 21'841 nem üres szinoníma halmaz
- 1'034'908 kép objektumaihoz van még határoló doboz annotáció is
- 1'000 szinoníma halmazhoz tartozik SIFT jellemzőkkel ellátott kép
- 1'200'000 kép van SIFT jellemzőkkel ellátva.

Látható hogy az előző három adathalmazt az IMAGENET már csak pusztta méreteivel is messze túlszárnyalja. Ezt mondhatjuk az etalon benchmarknak. Az évente megrendezett, a gépi látás "olimpiájának" számító ILSVRC(Large Scale Visual Recognition Challenge) is ezen az adatsokaságon szokott megrendezésre kerülni, jellemzően 4 kategóriában: objektum lokalizáció, objektum detekció, helyszín felismerés (pl tengerpart, hegyek), helyszín megértés. A legutóbbi nem takar kevesebbet mint egy kép szemantikus részekre való felosztása, például út, ég, ember vagy ágy. A [?] ábra ebből az adathalmazból mutat pár példát.

## A metrika

**MNIST és CIFAR** Az MNIST és a CIFAR-10/100 Adathalmazok esetében minden az egyszerű pontosság értéket nézzük, tehát az eltalált képek számát osztva a hibásan osztályozott képet számával.



**2.3. ábra.** Példa az IMAGENET adathalmaz képeire. Forrás: <http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2014/12/imagenet.jpg>.

**IMAGENET** Mivel az IMAGENET egy ennyire bonyolult adathalmaz, itt top 5 hibát szoktak nézni, ahol az számít sikeres találatnak ha a helyes címket a háló 5 legnagyobb valószínűséggel bíró tippjében benne van.

## 2.2. A Legelterjedtebb neurális hálózatok képfeldolgozáshoz

**Bevezetés** Az évek során számos neurális hálózattal kísérleteztek a kutatók annak érdekében hogy rájöjjönek melyek képesek legjobban megtanulni a képeken előforduló szabályosságokat, és ez alapján osztályozni őket. Ezekből szeretném a fő állomásokat kiemelni, és leírni hogy mik voltak a hiányosságok a meglévő architektúrákban amik új struktúrák létrehozását motiválták. Az áttekintésben nem ejtek szót a minden hálózat típusra érvényes általános, különféle reguralizációs eljárásokról, mint a súlyok felejtése vagy a dropout metódus. Csak a hálózatok felépítésének és tanításának architektúrális különbségeit veszem górcső alá.

### 2.2.1. A többrétegű perceptron (MLP)

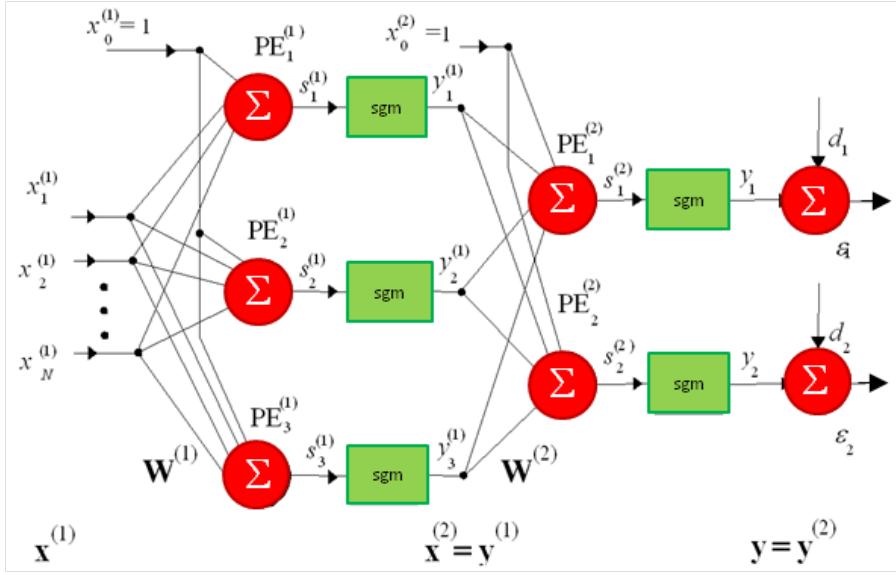
#### Az MLP előzményei és megalkotása

**Előzmények** Miután Rosenblatt megalkotta az első perceptron struktúrát a kései ötvenes években, a kutatók elkezdtek azon gondolkodni hogy hogyan lehetne ezeket a neuronokat összerendezni úgy, hogy együtt tanuljanak, és komplex regressziók, illetve osztályozási feladatok elvégzésére legyenek képesek. De ezek a kutatások sokáig igen meddőek voltak.

**Backpropagation** Az áttörés 1986-ban jött, amikor Geoffrey E. Hinton kollégáival sikeresen alkalmazta a hibavisszaterjesztéses algoritmust[23] az MLP súlyainak megváltoztatására a négyzetes hiba minimalizálásának érdekében. Az algoritmus lényege hogy a hibát a hálózatban a deriválás lánc szabályának segítségével terjesztjük vissza. Az algoritmust helymegtakarítás érdekében részletesebben nem ismertetem, az érdeklődők a bekezdés elején referált cikkben további részleteket találhatnak. Az algoritmus pseudokód összefoglalását a 2.4.-ábrán látható hálózat tanításához a 2.1.-lista mutatja.

#### Az MLP teljesítménye képosztályozási feladatokra

**Aktivitás a területen.** Gondolhatnánk hogy ezt a témát már rég elfejeztették a kutatók, de mivel az MLP egy igen egyszerű struktúra, ezért folyik még néhány kutatás hogy a határait megtalálják.



**2.4. ábra.** Egy kétrétegű MLP hálózat. Forrás: <https://mialmanach.mit.bme.hu/neuralis/ch04s01>

### 2.1. lista. A backpropagation algoritmus pseudokódja

```

inicializáljuk a háló súlyait (általában 0-1 közé eső véletlen számok)
do
    forEach tanító példa legyen tp
        jósolt_címke = háló-kiemelt(háló, tp)
        valódi_címke = tanító_címke(ex)
        hiba számítás f(jósolt_címke - valódi_címke) minden kimeneti egységen
        ΔW(2) kiszámítása
        ΔW(1) kiszámítása
        a hálózat súlyainak frissítése
    until Az összes bemenet sikeresen van osztályozva,
          vagy más megállási kritériumot el nem értünk
    return a hálózatot
  
```

**MNIST** Az MLP teljesítménye képosztályozási feladatok tekintetében igen szerény a többi hálózathoz képest, de az MNIST adathalmazzal még ez is egész jól megbírkózik, néhány figyelemre méltóbb eredményt a 4.5. táblázat foglal össze. A többi adathalmazon a naiv MLP nem hoz értékelhető eredményt, ennek az okait mindenkor megvizsgáljuk.

### 2.1. táblázat. Az MLP teljesítménye az MNIST adathalmazon

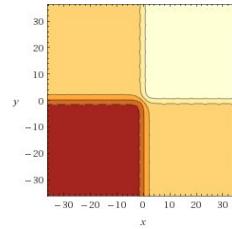
Rétegek száma	neuron struktúra	Teszt szet hiba százalék
3-réteg	768-300-10	4.7

### MLP hiányosságai

**A tanulás jellege** A felügyelt tanulás kapzsa módon a hibafüggvényt a súlyok gradiensének irányába optimalizálja, ezzel az a probléma hogy a hibafelület egy MLP esetében többé nem konvex mint egy egyszerű neuron esetében. A bonyolult hibafüggvény következtében lokális minimumok alakulnak ki. Sok esetben egy jó lokális minimumot sem érünk el naiv tanítással, a globális minimum elérésének az esélye pedig statisztikailag nulla. Ezt a jelenséget hivatott az alábbi egyszerű függvény  $\sigma^2(\sigma(x) + \sigma(y))$  kontúr diagrammja

(2.5. ábra). Ez habár nem közvetlenül egy hibafüggvény, de ezt a tulajdonságát jól szemlélteti.

**A struktúra kialakítása** Az MLP annyira általános struktúrát használ, hogy lényegében semmilyen a priori tudást nem használunk fel a hálózat tanításakor. Ez azt eredményezi hogy hatalmas kapacitás kell a képekből megjelenő bonyolult struktúrák megtanulásához, és a hálózat különböző részei kénytelenek rendre ugyanazt megtanulni. Sajnos az előbbi cél csak a hálózat növelésével érhető el, az MLP paraméter tere viszont nagyon rossz skálázódik. Ha veszünk egy 6 rétegű hálózatot aminek a rétegei rendre 2500-2000-1500-1000-500-10 neuronból állnak, és az MNIST esetén 784 elemű bemeneti vektorral rendelkezik, akkor optimalizálandó paraméter tér mérete annak folytán hogy minden réteg teljesen össze van kapcsolva már:  $784*2500 + 2500*2000 + 2000*1500 + 1500*1000 + 1000*500 + 500*10 = 11965000$ , ami már nyilvánvalóan hatalmas. Ez is tanítható sikeresen, de ahhoz már a továbbiakban bemutott kisegítő neurális struktúrák szükségesek.



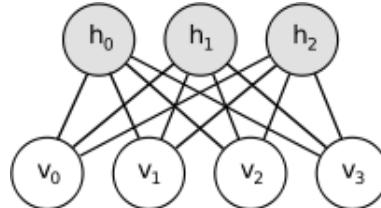
**2.5. ábra.** Példa egy komponált szigmoid függvényre.

### 2.2.2. A Korlátozott Boltzmann Gép

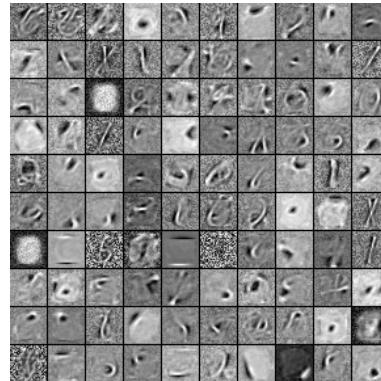
**Bevezetés** A korlátozott boltzmann gép (továbbiakban RBM - Restricted Boltzmann Machine, 2.6. ábra) egy sztochasztikus generatív neurális számítási modell. Működése az eddig tárgyalt MLP-től györekesen eltér, funkciója a bemenet jellemzőinek (featureinek) a megtanulása, és nem az egyes minták helyes osztályozása. Az intuitív jelentősége abban rejlik hogy feltehetjük hogy a naív MLP a kapzsi tanulása miatt nem képes megtanulni a minták valódi reprezentációját, de ha az MLP súlyait úgy tudnánk inicializálni, hogy a mintákban lévő szabályosságokat már eleve ismerje, akkor ebből könnyebben meg tudná tanulni hogy melyik jellemző mely osztályt azonosítja. Ezt felügyelt tanulás előtti fázist nem felügyelt előtanulásnak hívjuk. Szokás még erre a célra Autoencoder hálózatokat használni, illetve mélyebb hálókra az RBM és az Autoencoder többrétegű megfelelőit a mély hiedelem hálózatokat, és a "stacked" autoencodereket. Habár a legújabb eredmények szerint az Autoencoder hasonlóan jó eredményre vezet, és kevésbé bonyolult ezért gyakorlatban az ajánlott, én mégis az RBM-et választottam érdekes struktúrája miatt. Hugo Larochelle et al. hozott ki egy áttekintő tanulmányt az előtanulásról a mélyebben érdeklődőknek "Exploring Strategies for Training Deep Neural Networks" [8] címmel.

**Az RBM struktúrája** Az RBM mint említettem egy sztochasztikus generatív számítási modell, amelyben fontos hogy az egyes neuronok egy páros gráfot alkotnak (2.6. ábra). A generatív modell egy régi statisztikából származó fogalom ami azt foglalja magában

hogy a model képes megfigyelhető adatpontokat véletlenszerűen generálni. Az RBM egyik legtriviálisabb mérőszáma a rekonstrukciós hiba azt méri hogy ha egy adatpontot a háló bemenetére teszek, akkor azt milyen részletesen tudja visszagenerálni. Tehát az adatpont az a háló tanulási terében egy stabil pontnak számít-e. Ez a hiba mérték nem jó az RBM általánosító képességének mérésére, mégis sokan használják praktikus egyszerűsége miatt. Akit bővebben érdekel a téma a [16]-es referenciában talál bőséges irodalmat az RBM tanítását illetően. Itt csak az alapokra szorítkozok.



**2.6. ábra.** Egy RBM hálózat. Forrás: [http://deeplearning.net/tutorial/\\_images/rbm.png](http://deeplearning.net/tutorial/_images/rbm.png)



**2.7. ábra.** Egy RBM által megtanult filterek. Forrás: [http://www.pyimagesearch.com/wp-content/uploads/2014/06/rbm\\_filters.png](http://www.pyimagesearch.com/wp-content/uploads/2014/06/rbm_filters.png)

**Az RBM tanítása** Az RBM azért érdekes megközelítés a többi hálózathoz képest, mert probabilisztikus alapokon nyugszik. Az úgynevezett energia alapú hálózatok felfoghatóak úgy, hogy a hálózat minden konfigurációjához tartozik egy  $p(x)$  valószínűség, hogy mekkora valószínűsséggel tartózkodik a háló az adott konfigurációban. Azt szeretnénk elérni hogy az alacsony energiájú konfigurációknak nagy legyen a valószínűsége. Ez formalizálva a következő képpen néz ki:

$$p(x) = \frac{e^{-E(x)}}{Z} = \sum_h \frac{e^{(-E(x,h))}}{Z} \quad (2.1)$$

$$Z = \sum_x e^{-E(x)} \quad (2.2)$$

Ahol az  $E$  az energiafüggvényt jelenti. A fizikában jártasabb olvasók megfigyelhetik hogy ez a valószínűségi függvény megfelel a termodinamikában használt Boltzmann eloszlás valószínűségi függvényének. Az eredeti boltzmann gépet egy fizikus alkotta meg, pont erre az analógiára építve, azért hogy a Hopfield hálózatok gyengeségeit kiküszöbölje. A (2.1) képletet felhasználva megalkothatjuk a hibafüggvényünket, amely a negatív logaritmikus valószínűségi függvény (negative log likelihood function) lesz:

$$\mathcal{L}(\theta, x) = \frac{1}{N} \sum_{x^{(i)} \in \mathcal{D}} \log(p(x^{(i)})) \quad (2.3)$$

Definiáljuk a szintén a temodinamikából származó szabad energia függvényt:

$$\mathcal{F} = -\log \sum_h e^{-E(x,h)} \quad (2.4)$$

Ezzel újradefiniálhatjuk a valószínűségi függvényt:

$$p(x) = \frac{e^{-\mathcal{F}(x)}}{Z} \quad \text{ahol} \quad Z = \sum_x e^{-\mathcal{F}(x)} \quad (2.5)$$

Ami megengedi hogy a következőt írhassuk fel:

$$-\frac{\partial \log p(x)}{\partial \theta} \approx \frac{\partial \mathcal{F}(x)}{\partial \theta} - \frac{1}{|\mathcal{N}|} \sum_{\hat{x} \in \mathcal{N}} \frac{\partial \hat{x}}{\partial \theta} \quad (2.6)$$

Ha az előbbi függvényt deriváljuk, akkor megkapjuk a súlyváltozók update függvényét:

$$-\frac{\partial \log p(v)}{\partial W_{ij}} = E_v[p(h_i|v) * v_j] - v_j^{(i)} * \text{sigm}(W_i * v^{(i)} + c_i) \quad (2.7)$$

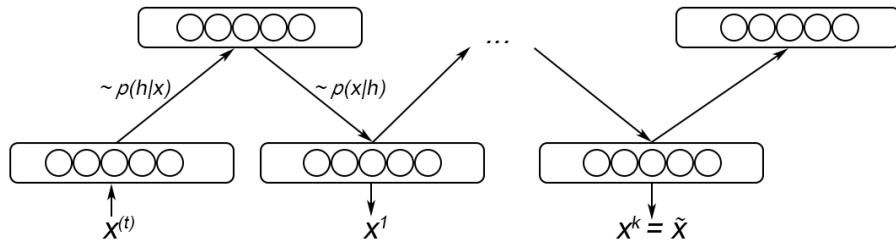
$$-\frac{\partial \log p(v)}{\partial c_i} = E_v[p(h_i|v)] - \text{sigm}(W_i * v^{(i)}) \quad (2.8)$$

$$-\frac{\partial \log p(v)}{\partial b_j} = E_v[p(h_i|v) * v_j] - v_j^{(i)} \quad (2.9)$$

Ebből látható hogy az egyes deriváltak kiszámításához szükségünk van a rejtett neuronok valószínűségére a bemeneti neuronok állapotától függően. Ezért szükséges hogy a boltzman gép korlátozott legyen, tehát egy páros gráf formáját vegye fel, mert így az egyes neuronokhoz tartozó valószínűségek párhuzamosítva számolhatóak, mivel függetlenek egymástól. Erre egy gyors eljárás a kontrasztív divergencia algoritmus amelynek a valószínűségi mintavételét a 2.8. ábra szemlélteti. A gibbs mintavételezéshez tartozó markov lánc lépéseiinek a képleteit a (2.10) és a (2.11) képlet írja le.

$$h^{n+1} \sim \text{sigm}(W^T v(n) + c) \quad (2.10)$$

$$v^{n+1} \sim \text{sigm}(W^T h^{(n)} + b) \quad (2.11)$$



**2.8. ábra.** A gibbs mintavételezési eljárás. Forrás: <http://recognize-speech.com/images/nicolas/Gibbs.png>

Ahol  $c$  és  $b$  az eltolás súlyvektorokat jelzik.

**Az RBM és DBN előtanulás eredményei** Az hogy a hálót előtanítjuk fantasztikus eredményjavulást hozott, amely beigazolta hogy valóban a számok valós jellemző reprezentációihoz közel helyezkedik el egy nagyon optimális lokális minimum. Az 2.2. táblázat szemlélteti az eredményeit az MNIST adathalmazon. Látszik hogy ennek segítségével jóval nagyobb hálók képezhetőek ki és lényeges jobb eredményre vezetnek.

**2.2. táblázat.** Az MLP teljesítménye az MNIST adathalmazon RBM előtanulást használva. Forrás: [2]

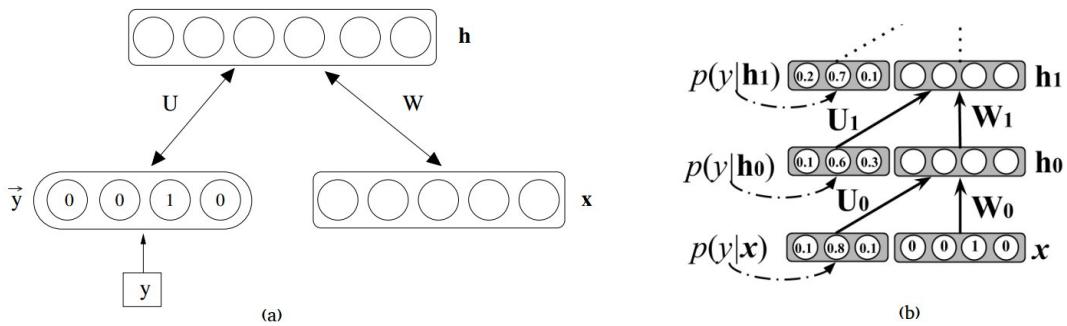
Rétegek száma	neuron struktúra	Teszt szet hiba százalék
3-réteg	768-800-10	0.7
6-réteg	784-2500-2000-1500-1000-500-10	0.35

**További módszerek az RBM használatára osztályozáshoz** Az előtanulásban nem merül ki az RBM jellegű hálózatok potenciálja, ha osztályozási feldatokról van szó, egy pár további lehetőséget szeretnék még érintőlegesen megemlíteni.

- Egy másik, lehetőség a szerint az egyes osztályoknak egy RBM-et hozunk létre, és egy Softmax modell-t tanítunk a szabad energia függvényükön. A partiós függvényt ebben az esetben a softmax paramétereit approximálják.[16]
- Egy harmadik lehetőség hogy két külön látható réteg csoportot tartunk, egyet a kép adatoknak, egy másikat pedig a tanító címkeknek, ebben az esetben az osztályhoz tartozási valószínűséget a teszt vektor, és a címkek szabad energiájának eloszlásában határozzunk meg. Tehát minnél alacsonyabb lesz a szabad energia egy címkelvel, annál valószínűbb hogy a teszt vektor abba az osztályba tartozik, itt végülis tanulásnál a címkek és a mintapontok együttes valószínűségi sűrűségfüggvényét becsüljük.[16]
- Egy újabb fejlemény Hugo Larochelle Hibrid RBM [18] struktúrája, amit közvetlen az RBM-en belül kombinálja a generatív és a diszkriminatív modellek előnyeit, és

egyszerre tud tanulni on-line jelleggel címkezett és címkezetlen adatokból egyaránt. A struktúrát a 2.9. ábra szemlélteti.

- A Larochelle féle struktúra továbbfejlesztését szintén a 2.9. ábra szemlélteti. Ez az úgynevezett "Stacked Boltzmann Experts Network"[7]. Itt minden szint ad egy valószínűséget a tanító minta osztályzására, és ezeknek az átlagolásával születik meg a végeges predikció. Ezek a hálózatok azért végtelenül izgalmasak, mert habár ha sok a tanító minta, akkor ugyanolyan a teljesítménye mint a klasszikus módszereknek, de ha kevés címkezet adat áll rendelkezésre, akkor vastagon jobb teljesítményt hoz, mitn ahogyan az a 2.10. ábra is mutatja. Érdemes megemlíteni hogy a publikációban újságcikkekben, és nem képeken tesztelték.



**2.9. ábra.** A (a) Larochelle féle hibrid RBM [18] és (b) annak a többszintű továbbfejlesztése az SBEN[7] struktúra.

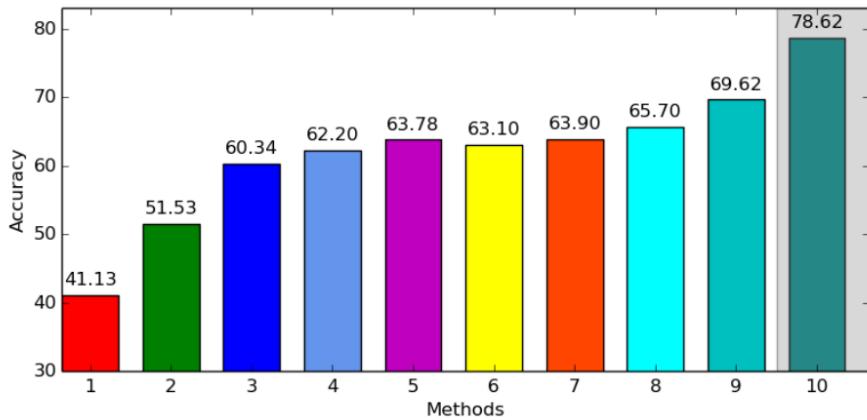
	Error	Precision	Recall	F1-Score
<i>NB-EM</i>	$0.369 \pm 0.039$	$0.684 \pm 0.022$	$0.680 \pm 0.028$	$0.625 \pm 0.043$
<i>MaxEnt-ST</i>	$0.402 \pm 0.026$	$0.623 \pm 0.025$	$0.593 \pm 0.015$	$0.583 \pm 0.020$
<i>SVM-ST</i>	$0.342 \pm 0.020$	$0.663 \pm 0.010$	$0.665 \pm 0.014$	$0.644 \pm 0.015$
<i>HRBM</i>	$0.252 \pm 0.023$	$0.740 \pm 0.019$	$0.765 \pm 0.016$	$0.741 \pm 0.021$
<i>3-Rect</i>	$0.328 \pm 0.020$	$0.673 \pm 0.017$	$0.680 \pm 0.021$	$0.654 \pm 0.023$
<i>3-SBEN,BU</i>	$0.239 \pm 0.015$	$0.754 \pm 0.014$	$0.780 \pm 0.016$	$0.754 \pm 0.015$
<b><i>3-SBEN,BUTD</i></b>	<b><math>0.210 \pm 0.011</math></b>	<b><math>0.786 \pm 0.009</math></b>	<b><math>0.784 \pm 0.014</math></b>	<b><math>0.777 \pm 0.012</math></b>

**2.10. ábra.** Az SBEN[7] és más, klasszikus struktúrák eredményei a WEBKB adathalmazon, úgy hogy csak a tanító adatok 1%-a volt felcímkezve (8 példa osztályonként). A mérés teljes riportját az SBEN publikációban[7] lehet megtalálni.

### 2.2.3. State of the art MLP hálózatok

**További kutatások** Az MLP hálózatokat a mai napig nagy érdeklődés övezi egyszerű struktúrájuk miatt. Látszik hogy az MNIST adathalmazon már nem nagyon van hova javítani az MLP-k teljesítményét, viszony mint azt pár paragrafussal előbb megemlítettem a paraméter tér csökkentésében, és komplexebb adathalmazokhoz még van fejleszteni való ezeken a struktúrákon. Egy igen friss publikáció amelynek címe "How far can we go without convolution: Improving fully-connected networks"[24] arra mutat rá hogy hogyan lehet az

MLP hálózatok paraméter terét úgy csökkenteni hogy a sigmoid rétegek közé kis méretű lineáris rétegeket teszünk be, példának okáért legyen a két réteg 1500-2000 neuron, akkor a teljes paraméter terük  $1500 * 2000 = 3000000$ , de ha közé teszünk egy 500 neuronos lineáris rétege, akkor ez lecsökken  $150 * 500 + 2000 * 500 = 1075000$  paraméterre, ami igen szignifikáns redukciót jelent a hálózat komplexításában. Ez a redukció oly mértékű hogy a fentebb említett publikációban vizsgált legnagyobb struktúra paraméter terét 112 millióról 2.5 millióra csökkentették, összehasonlítás képpen egy modern konvoluciós hálónak 3.5 millió paramétere van. Látható hogy ezzel sikerült a kutatóknak megoldania a többrétegű perceptron gépek egyik legnagyobb problémáját, a mértéktelenül burjánzó paraméter teret. Ezen felül az eltünő gradiensek problémáját is orvosolja, amivel itt bővebben nem foglalkozunk. Viszont az eredményeit a CIFAR-10 2.11. ábra szemlélteti. Az előbb említett táblázat nagyon jól összefoglalja az MLP-k teljesítményének a fejlődését a CIFAR-10es adathalmazt használva.



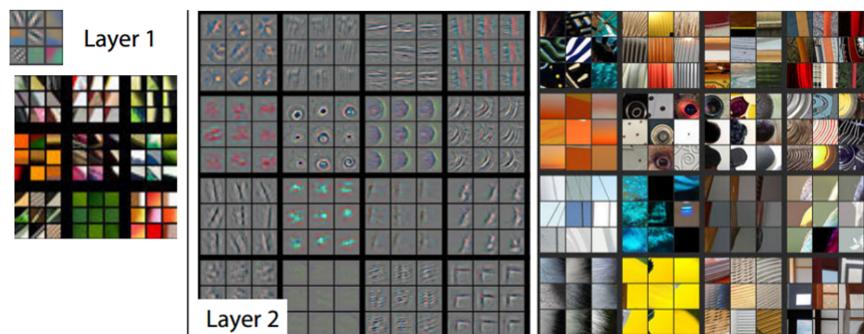
**2.11. ábra.** Az MLP fejlődése a CIFAR-10 adathalmazon. (1) Logiszikus regresszió fehérített adatokon; (2) Tiszta backpropagation egy 782-10000-10 méretű hálózaton; (3) Tiszta backpropagation egy 782-10000-10000-10 méretű hálózaton. (4) Egy 10000-10000-10es hálózaton, RBM előtanítással, az utolsó réteg logiszikus regresszió; (5) Egyrétegű 10000 neuronos hálózat logiszikus regressziós kimenettel, RBM előtanítással; (6) "Fastfood FFT" model (7) Zerobias autoencoder hálózat 4000 rejtett neuronnal és logiszikus regressziós kimenettel; (8) 782-4000-1000-4000-10 Z-Lin hálózat; (9) 782-4000-1000-4000-1000-4000-1000-4000-10 Z-Lin hálózat dropoutokkal; (10) Ugyanaz mint a (8), csak adat augmentációval. Az (1)-(5) eredmények Krizhevsky és Hinton 2009-es publikációjából származnak. A legutolsó azért szürkített, mert adat augmentációt használ. Forrás: [24]

## 2.2.4. Konvoluciós hálózatok

**Bevezetés** A szakdolgozatom harmadik nagy részét a konvoluciós hálózatok teszik ki. Miután az irodalomkutatásom közben rá kellett jönnöm hogy az általam tanult klasszikus MLP struktúrák nem képesek a komplex jelek, mint például az MNIST adathalmaznál összetettebb képek kielégítő megtanulására, kénytelen voltam új irányok után nézni. Így találtam meg a konvoluciós architektúrákat. Ezek korunk legjobban teljesítő architektúrái,

ennek oka hogy a jelek nagy része amit fel szeretnénk dolgozni az emberi érzékszervek által érzékelt jelek - például képek - amik jelentős transzlációs invarianciával rendelkeznek, és ezek a hálózatok ezt az a priori tudást beleépítik az architektúrába, így lényegesen kevesebb paramétert kell megtanulnunk mint egy *elméleti síkon* hasonló teljesítményű MLP-nél. Mivel természetesen tudjuk Cybenko (1989)[15] és Kurt Hornik (1991)[17] publikációi után hogy egy kétrétegű MLP-vel tetszőleges függvényt képes approximálni, de ehhez annyi neuron kellene a komplex jelek esetében mint a képek hogy praktikusan nem kivitelezhetőek ezek a hálózatok. A rétegek növelésével és hálózati kényszerek bevezetésével ez a paraméter tér jelentősen csökkenhető.

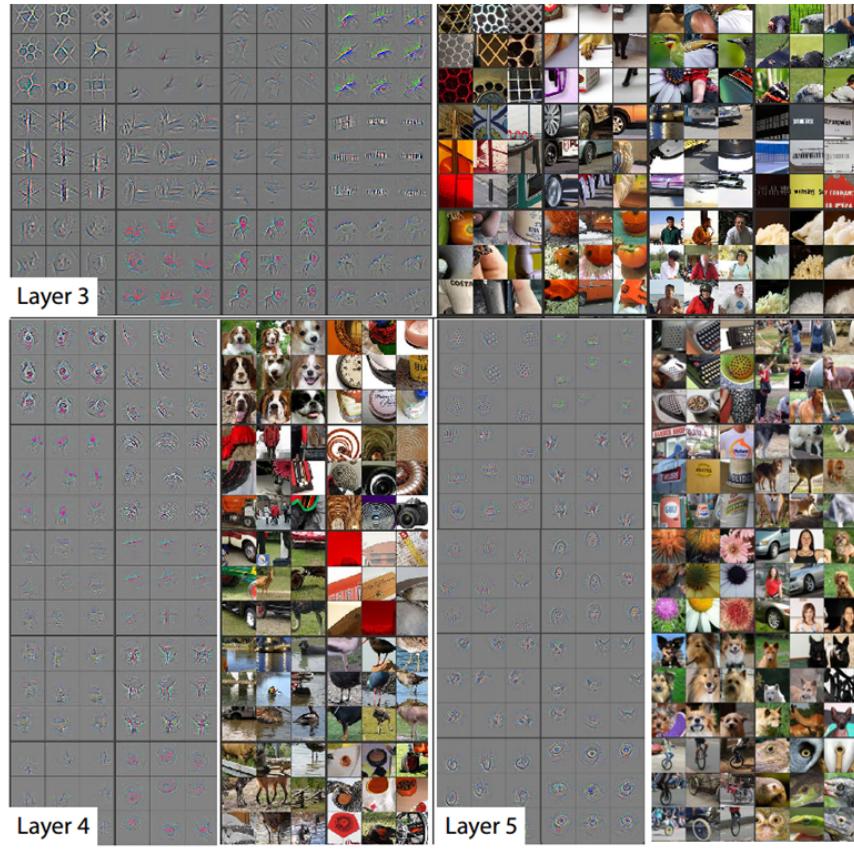
**Az intuició** A konvoluciós architektúra teljes mértékben biológiaiag inspirált, a macska vizuális kortexének a feltérképezésénél találtak hasonló kapcsolatokat az állat agyában[5], és ennek a mintájára építették fel a mesterséges hálózatot. Alapötlete hogy magába a hálózati struktúrába foglaljuk bele a jel transzlációs invarianciát. A modellt eleinte képek feldolgozására alkották meg, és az előbbi mondat itt is szemléltethető a legintuitivabban. Tegyük fel hogy van egy képünk amin vagy egy objektum, akkor ha fel kell ismerni hogy a kép az adott objektumnak a jellemzőit tartalmazza-e akkor nekünk adott esetben ugyan annyi információval szolgál ha ez a jellemző (mondjuk egy sarok) a bal vagy a jobb oldalon van a képen. Természetesen ezek a lokális struktúrák a rétegekkel felfele egyre globálisabbak lesznek, az egyre magasabb szinteken pedig több jellemzőből komponált összetett jellemzők jelennek meg. És a hálózat az összetett jellemzők jelenlétéből következtet a kép osztályára. Ezt hivatott szemléltetni a 2.12. ábra és 2.13. ábra amely egy konvoluciós háló egyes rétegeinek a szűrőit mutatja be, és hogy milyen képelemek aktiválták őket a leginkább. Az első sikeres alkalmazása ennek a modellnek a LeNet[14] volt 1990-ben, amelyet irányítószámok, karakterek és hasonló dolgok felismerésére használtak, de a modell sokáig nem kapott nagy érdeklődést.



Visualizations of Layer 1 and 2. Each layer illustrates 2 pictures, one which shows the filters themselves and one that shows what part of the image are most strongly activated by the given filter. For example, in the space labeled Layer 2, we have representations of the 16 different filters (on the left)

**2.12. ábra.** Az első két szint szűrői egy konvoluciós hálózatban. Forrás: "Visualizing and Understanding Convolutional Neural Networks" [21]

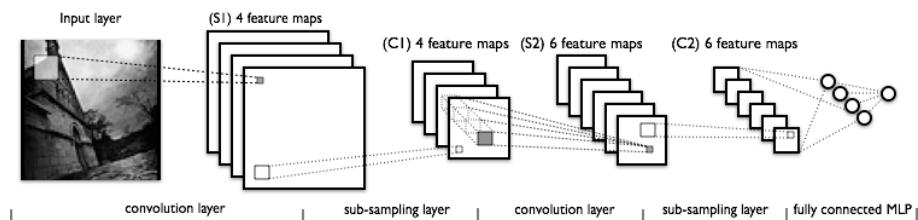
**Matematikai interpretációja** Tegyük fel hogy van egy  $32 \times 32 \times 3$ -as képünk. Ahhoz hogy egy teljes rétegbe kapcsoljuk bele mondjuk 1024 neuronnal ki kellene lapítanunk, és



Visualizations of Layers 3, 4, and 5

**2.13. ábra.** Az felsőbb rétegek szűrői egy konvoluciós hálózatban. Forrás: "Visualizing and Understanding Convolutional Neural Networks" [21]

egy  $32 * 32 * 3 * 1024 = 3'145'728$  paraméterünk lenne az első rétegben. De tegyük fel hogy a legkisesebb jellemző amit érzékelni akarunk az egy  $3 \times 3$  méretű patchen van a képen, viszont akárhol lehet, akkor ha az első rétegben 32 jellemzőt szeretnénk érzékelni, akkor csak  $3 * 3 * 32 = 288$  paraméterre lesz szükségünk az első rétegben, ami jelentős redukció. Ezután a következő réteg ehhez a 288 neuronhoz fog kapcsolódni, és ha ott 64 neuron lesz akkor  $3 * 3 * 64 = 567$  neuronra kell majd, amik az eredeti képből viszont már egy  $5 \times 5$ -ös szeletet fognak indirekt módon lefedni. Ezt mutatja be egy klasszikus architektúra a leNet a 2.14. ábrán.



**2.14. ábra.** Az felsőbb rétegek szűrői egy konvoluciós hálózatban. Forrás: "Visualizing and Understanding Convolutional Neural Networks" [21]

**Az IMAGNET** 2012-ben az AlexNet nevű konvoluciós hálózat amelyet Alex Krizhevsky, Ilya Sutskever és Geoffrey Hinton alkottak főlénnyel megnyerte az azévi ILSVRC versenyt. A háló top 5 hibája (az olvasó konzultáljon az adathalmazokat bemutató résszel a metrika leírásáért.) 16% volt, míg a második helye ami egy SVM-eket használó modell volt 26%-os hibát produkált. Ez a 10%-os különbség az egekbe emelte a konvoluciós hálózatok népszerűségét, és hivatalosan is elhozta a neurális képfeldolgozás korát. Innen előt kezdve minden évben konvoluciós hálózatok nyerték meg az ILSVRC-t. A 2.3. táblázat bemutatja az egyes évek eredményeit a hálót néhány paraméterével együtt. Összehasonlítás képpen, egy átlagos ember teljesítménye az adathalmazon 5-10 hibaszázalék körül mozog.

### 2.3. táblázat. Az ILSVRC győztesei

Év	A struktúra neve	Tanítás ideje	Paraméter tér mérete	Top 5 hiba százalék
2012	AlexNet [6]	két GTX 580 GPU-n 5-6 nap	60 millió	16%
2013	ZF Net [21]	egy GTX 580 GPU-n 12 nap	60 millió	11.2%
2014	GoogLeNet [13]	"néhány high end GPU-n egy héten belül"	4 millió	6.7%
2015	Microsoft ResNet [10]	8 GPU-s gépen 2-3 hétag	N\A	3.6%

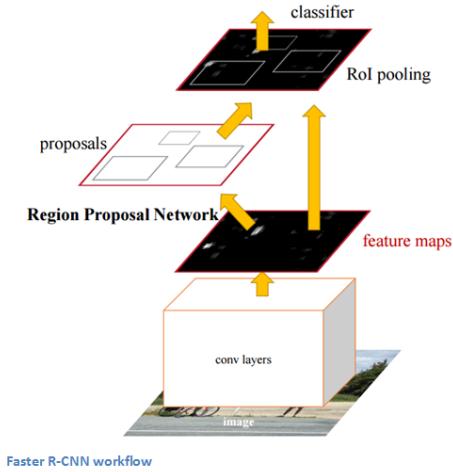
**Értékelés** Látható hogy a legújabb konvoluciós architektúrák már az emberi kiértékelésnél is pontosabb eredményt hoznak. Ráadásul a kezdeti naív megközelítést követően a paraméter tér is drasztikus csökkenésnek indult. Manapság ha valaki képosztályozási feladatot szeretne végezni neurális hálózatokkal, akkor egy ilyen előre elkészített architektúrát fog használni. Sajnos az is jól nyomon követhető hogy a hálózatok fejlődésével a szükséges hardware kapacitás is meredek emelkedésnek indult. Ha az ember egy konvoluciós architektúrát egy saját adathalmazra szeretne megtanítani akkor komoly infrastruktúrával kell rendelkeznie hozzá. Éppen itt domborodik ki a tensorflownak a dolgozat elején említett előnye, hogy miután pythonban specifikáltuk a struktúrát azt képesek vagyunk minden erőfeszítés nélkül egy 8 GPU-s fürtre szétterjeszteni és tanítani, majd utána a paraméter teret lementve akár egy mobilon a megtanított hálót újra betölteni és akár valós idejű inferenciát futtatni. A paraméter tér ha 16 bites floatokkal számol az ember akkor 4 millió paraméternél kb 8 megabyte lesz, ami még egy igen kezelhető mennyiség.

## 2.3. További érdekes irányok a neurális képfeldolgozásban

Itt, az irodalom kutatásom tárgyalásának végén elérünk arra a pontra ahol éppen a tudomány tart, ebben a fejezetben röviden fel szeretném villantani a neurális képfeldolgozás legújabb és legérdekesebb irányait.

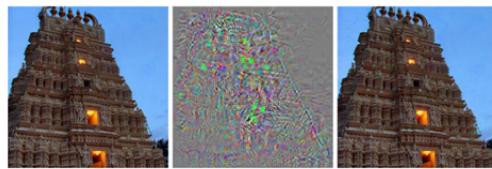
**Régió alapú konvoluciós hálózatok[12]** Sokan azt mondják hogy ez a publikáció csokor (R-CNN, Fast R-CNN, Faster R-CNN) hosszú idők óta az egyik legfontosabb amit új neurális architektúrákról olvashatott az ember. Eddig meg tudtuk mondani egy hálóval hogy tartalmaz-e a kép valamelyen objektumot. Az R-CNN hálózatok már az objektum pontos helyét is megmondják a képen, ami egy minőségbeli ugrást jelent. Az 2.15. ábra szemlélteti a módszert. A módszer lényege hogy a feladat két neurális hálózatra

van faktorizálva amik tandemben dolgoznak, az egyik egy osztály agnosztikus objektum detektor, míg a másik egy osztályozó hálózat.



**2.15. ábra.** A Faster R-CNN munkafolyamata

**Generatív adverziális hálózatok[9]** A LeCunn, a konvoluciós hálók megalkotója szerint ez az utóbbi 10 év legérdekesebb ötlete a területen. A lényeg hogy két hálózatot tanítunk tandemben, egy generatív és egy diszkriminatív modell-t. A diszkriminatív modell dolga edönten egy képről hogy valódi-e vagy sem, a generatívé pedig hogy olyan képeket tudjon generálni amivel átveri a másik modell-t, ezért hívják adverziális hálózatnak. Az egész súlya abban rejlik hogy így a diszkriminatív hálózatnak meg kell tanulni az adat egy nagyon jó reprezentációját hogy képes legyen dönteni, ezzel mintegy nem felügyelt módon a legfontosabb jellemzőket kiemelni a képből. A generátor a végére pedig képes lesz valósághű képeket "álmودni". A 2.16.-ábra mutat egy tipikus tanító példát.



**2.16. ábra.** Jobbra: Eredeti kép, középen: Perturbációk, balra: Perturbált kép. A jobb oldalit helyesen, a bal-t pedig hibásan osztályozná egy CNN.

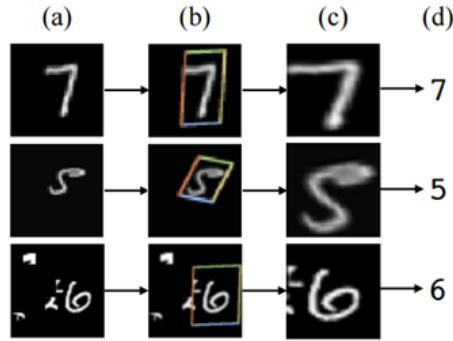
**Képleírások generálása[4]** A nem olyan távoli múltban nagyon sok érdekes publikáció jelent meg olyan neurális struktúrákról amelyek képek leírására alkalmasak. Lényegében egy CNN és egy RNN hálózat működik együtt, és fantasztikus dolgokra képesek. Tovább nem is taglalnám, mert nagyon sok előismeretet igényel a téma, az érdeklődők a megfelelő publikációt megtalálják az irodalomjegyzékben. A 2.17. mutatja az eredményt.

**Térbeli transzformációs hálózatok (STN [11])** A fejlesztés jelentősége abban rejlik hogy eddig mindig vagy a háló struktúrájába kellett belekódolni ha valamilyen variancia



**2.17. ábra.** A Faster R-CNN munkafolyamata

ellen védeni szerettük volna a hálózatot, vagy pedig az adathalmazt kellett úgy augmentálni hogy a háló jól általánosítson. Az előbbire egy jó példa a CNN hálózatok max-pooling rétege, az utóbbit pedig az hogy mondjuk minden képet elforgatva is beadunk a háló tanításakor, hogy invariáns legyen rotációra a megtanult modell. Az STN hálózatok mint egy modulként kapcsolhatóak az egyes hálózatok elé, és ezeket a problémákat megfelelő tanítás után automatikusan megoldják. Jól látható hogy a neurális fejlesztés a monolitikus hálózatokból szintén elkezdett a modulokból felépülő paradigmája felé menni. Az 2.18. ábra mutat a hálózat működésére egy példát.



**2.18. ábra.** Egy térbeli transzformációs hálózat lépései.

**irodalomkutatás összefoglalása** Ezzel az irodalom kutatásom végére értem. Úgy vélem hogy a neurális képosztályozás legtöbb fontos architektúrájára kitért, és kaptam egy átfogó képet arról hogy hol tart a terület, és milyen módszerekkel érdemes nekiállni egy ilyen problémának. A továbbiakban szeretném bemutatni a saját munkámat, hogy mely hálózatokat implementáltam, mértem le a saját rendszerem, és ebből milyen tanulságokat vontam le.

### 3. fejezet

## Hálózatok impelmentálása és elemzése tensorflowban

A következő részben szeretném bemutatni saját mukámat és mérési eredményeimet. Az előző részben bemutatott hálózatfajtából megvalósítottam számtalan példányt tensorflowban, és méréseket végeztem rajtuk az MNIST és a CIFAR-10 adathalmazon. A munkám célja az volt hogy letesztem a tensorflow lehetőségeit kísérleti hálózatok kifejlesztésére és monitorozására. A fejezet a következő részekre tagolható:

1. A baseline metódusok bemutatása.
2. A fejlesztési környezet bemutatása.
3. Az általam használt optimizátorok ismertetése.
4. Saját fejlesztések bemutatása.

#### 3.1. A baseline osztályozók

Természetesen nem lehet méréseket végezni referencia adatok nélkül, ezért a CIFAR-10 és az MNIST adathalmazon is lefuttattam két ismert, minden nehézség nélkül használható osztályozó algoritmust. Az egyik a Logisztikus regresszió volt, a másik pedig az SVM. Mindkettő bemenetére közvetlen a kép pixeljeit tettek.

#### 3.2. A saját magam által kialakított fejlesztőkörnyezet

A Tensorflow ökoszisztemája nagyon jó pontja a csúcsra járatott monitorozási lehetőségek, viszont nem triviális egy olyan struktúra kialakítása ahol az ember nagy hatékonysággal dolgozhat. Hosszas kísérletezés után a következő munkafolyamatot találtam a legjobbnak:

- *Gyors prototipizálás:* Erre a célra a Jupyter notebookokba írt TF-Slim magas szintű API-t találtam a legjobbnak, így nagyon gyorsan le lehet akármilyen ötletet tesztelni és kiértékelni.

- *Stabil modellek fejlesztése:* Ha egy modell túljutott a pár soros méreten, vagy tényleg egy nagyobb rendszer részeként szeretnénk használni akkor azt érdemesnek találtam osztályba foglalni, és a fejlesztéshez PyCharm IDE-t használni mert lényegesen gyorsabban lehet vele haladni komplex python kódnál mint a notebookokkal.
- *A modellek kiértékelése:* A modellek kiértékelésére úgy gondolom hogy a Tensorflowval érkező Tensorboard a legalkalmasabb, mint majd látni fogja az olvasó a modelleim elemzésénél hogy ez az eszköz lehetővé teszi komplex struktúrák elemzését egészen a tanulástól az igényelt rendszer erőforrásokig.
- *A modellből kinyert adatok részletes elemzése:* Erre a célra az klasszikus tudományos csomagok használatát találtam a legjobbnak jupyter notebookban alkalmazva, mert így egy helyen van a modell futtatása, a mérési eredmények és azt elemző kód.
- *Egzotikusabb modellek kivitelezése:* Ha az ember olyan modelleket szeretne kódolni amik túlnyúlnak a klasszikus struktúrákon, akkor kénytelen lenyűlni a Tensorflow eredeti programozási absztrakciójához, mint ahogyan azt az RBM esetében látni fogjuk. Ez az alacsony API hatalmas szabadságot ad a programozónak, de iszonyatosan bőszavú (verbose).

### 3.3. A kísérletek alatt használt optimizátorok

Nagyon sok optimizátor létezik a numerikus optimalizálási feladatok megoldására, ezt most nem is szeretném különösebben taglalni mert ez egy olyan terület amiből már sok szép Phd. munka született és hozzávetőlegesen 20-30 oldallal növelné meg a dolgozatomat. Dióhéjjban, én két optimizátorral kísérleteztem, a klasszikus *sztochasztikus mini-batch* és az *adaptív momentum becslési* módszerrel. Az első előnye hogy kevesebbet kell számolni, de vagy konstans tanulási rátát használ, vagy kívülről kell valamilyen lehűléses algoritmussal kontrollálni ezt, ami nehezebbé teszi a használatát. A második egy adaptív módszer, ahol az optimizátorba bele van építve a tanulási ráta beállítása, így dinamikusan tudja lekezelni a hibafelület topológiájának a változását. Ennek az algoritmusnak két hiperparamétere van, de ezeket nem feltétlen kell megadni, mert a módszer szerző mindkettőhöz megadtak általánosságban jól működő értékeket. A különöző optimizátorok összehasonlításáról az érdeklődők <http://sebastianruder.com/optimizing-gradient-descent> ezen a honlapon olvashatnak egy nagyon jó összefoglalást.

### 3.4. A saját implementációk bemutatása

#### 3.4.1. A hálózatok monitorozása

A Tensorflow érett API-t kínál a hálózat paramétereinek követésére tanulás közben, de nem ment le automatikusan minden egyes lefutáskor minden változó értékét, mivel ez egy modern hálózatnál több millió értéket is jelenthet. A VGG konvoluciós háló például 140 millió paramétert tartalmaz. A hálózat változóiról általánosságban a következő értékeket

mentettem le: minimum érték, maximum érték, közép érték és a standardizált szórásukat. hogy ezeket ne kelljen minden változóra kiadni, ezért a 3.1.-es függvényt alkalmaztam.

### 3.1. lista. A változók mentése

```
def variable_summaries(name, var):
    """Attach a lot of summaries to a Tensor."""
    with tf.name_scope('summaries'):
        mean = tf.reduce_mean(var)
        tf.scalar_summary('mean/' + name, mean)
        with tf.name_scope('stddev'):
            stddev = tf.sqrt(tf.reduce_sum(tf.square(var - mean)))
        tf.scalar_summary('sttdev/' + name, stddev)
        tf.scalar_summary('max/' + name, tf.reduce_max(var))
        tf.scalar_summary('min/' + name, tf.reduce_min(var))
        tf.histogram_summary(name, var)
```

### 3.4.2. A logiszikus regresszió

Az első modell amit implementáltam Tensorflowban az a logisztikus regresszió volt. A célja az volt hogy összehasonlítsam a Tensorflow erőforrás igényét egy klasszikus python machine learning csomag, a Scikit-learn igényeivel. A struktúrát az <szám> ábra szemlélteti. Egyből látszódhat hogy az avatatlan szem számára a tensorflowboard eléggé nehezen értelmezhető lehet, ezért is szokták mondani hogy a tensorflownak a tanulási görbéje viszonylag nagy. Szerencse hogy lehet benne névtereket létrehozni hogy a gráfok könnyebben átláthatóak legyenek. Az egész tanulási gráfot a járulékos nodeokkal az <szám> ábra szemlélteti.

### 3.4.3. A többrétegű perceptron

A többrétegű perceptronnal való kísérletezést a TF-Slim API nagyon megkönnyíti, minden nehézség nélkül a rétegeket egymás után tenni, ha pedig egyedi elemet szeretne az ember definiálni akkor arra is lehetőség van. Hasonló architektúrákat teszteltem az MNIST és a CIFAR-10 adathalmazon is, ami nagyon jól megfogta a két adathalmaz közötti különbséget. A 3.2.-listázás egy ilyen háló definicióját szemlélteti.

Mint látható nagyon könnyen állíthatóak a regularizációs tagok, megadható több félet optimalizáló és hibafüggvény is, ami igazán könnyűvé tette a kísérletezést.

Az MLP-ben használt elemi alkotóelemeim:

- *fc*: A teljesen kapcsolt réteg, minden neuron, minden előbbi réteg neuronjával össze van kötve, ennek a definicióját a (3.1)-képlet mutatja be, ahol  $l$  a réteg sorszáma, és "Activation" egy tetszőleges aktivációs függvény.
- *relu*: A rektifikált lineáris egység (ReLU) egy aktivációs függvény, ezeket a teljesen kapcsolt réteg után kapcsoljuk, azért hogy nem-linearitásokat vigyük a rendszerbe. így növelve a leképző képességét. A ReLU definíciója a (3.2) képleten látható.
- *sgm*: A sigmoid aktivációs függvény amely szintén egy aktivációs függvény mint a ReLU, de nehezebb a deriváltját számolni, és érzékenyebb az adatok normalizáltságára. Az sigmoid definíciója (3.3) képleten látszik.

### 3.2. lista. Egy MLP definíciója

```

def fully_connected(batch_data, batch_labels):
    with slim.arg_scope([slim.fully_connected],
                        activation_fn=tf.nn.relu,
                        weights_initializer=tf.truncated_normal_initializer(0.0, 0.01),
                        weights_regularizer=slim.l2_regularizer(0.0005)):
        # First Layer
        x = slim.fully_connected(batch_data, 400, scope='fc/fc_1')
        variable_summaries('fc/fc_1', x)

        # Second Layer
        x = slim.fully_connected(x, 1024, scope='fc/fc_2')
        variable_summaries('fc/fc_2', x)

        # Third Layer
        last_layer = slim.fully_connected(x, 10, activation_fn=None, scope='fc/fc_3')
        variable_summaries('fc/fc_3', x)
        predictions = tf.nn.softmax(x)

        slim.losses.softmax_cross_entropy(last_layer, batch_labels)
        total_loss = slim.losses.get_total_loss()
        tf.scalar_summary('losses/total_loss', total_loss)

    optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
    return optimizer, predictions

```

$$X^{(l)} = W * Activation(X^{(l-1)}) \quad (3.1)$$

$$reLu(x) = max(0, x) \quad (3.2)$$

$$sgm(x) = \frac{1}{1 + e^x} \quad (3.3)$$

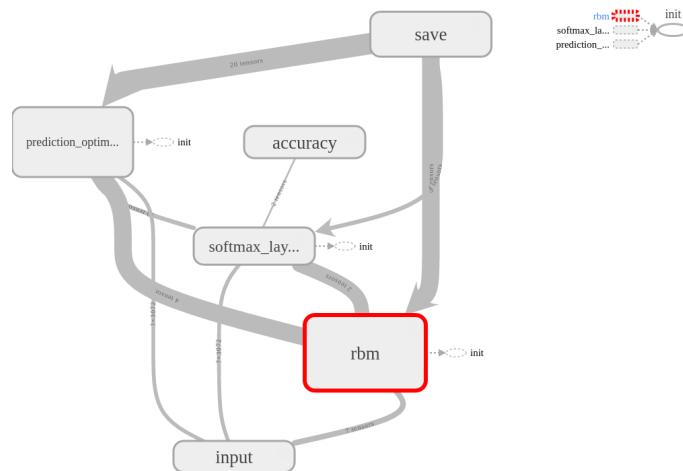
#### 3.4.4. Az RBM hálózat

Mint mondtam a TF-Slim nagyon kényelmes volt addig amíg olyan struktúrákkal dolgoztam amik könnyen definiálhatóak. Viszont csak többrétegű perceptronok és konvoluciós hálózatokat lehet benne egyelőre alkotni. Az korlátozott boltzmann gép implementálásához le kellett mennem a Tensorflow alacsony szintű interfacéhez amiben a hálózat implementálása több mint egy hét volt. Viszont ezalatt értettem meg igazán hogy hogyan működik egyszerűbb a könyvtár, másrész pedig az RBM-ek. Az RBM struktúra legtrükkösebb része a kontrasztív divergencia volt, mivel az egy valószínűségi döntés, de az egyes batchek futása közben nem tudok közvetlenül a gráfban belül véletlen számokat generálni változtatható mennyiségben. Azért nem lehet egy adott méretben generálni, mert a batch méret változik, és minden számnak kellett generálni. A megoldás amit alkalmaztam az az volt hogy numpy-ban legeneráltam minden tanításnál egy akkora mátrixot véletlen számokból mint amekkora a tanító halmazom volt. Majd a bináris 0-1 döntést a következő képpen szimuláltam:

1. Kivontam a valószínűségi mátrixot a random számokból

2. Vettem az eljöjelét a keletkezett mátrixnak
3. Átvezettem egy relu rétegen, amitől 0 és 1 közé normalizálódott.

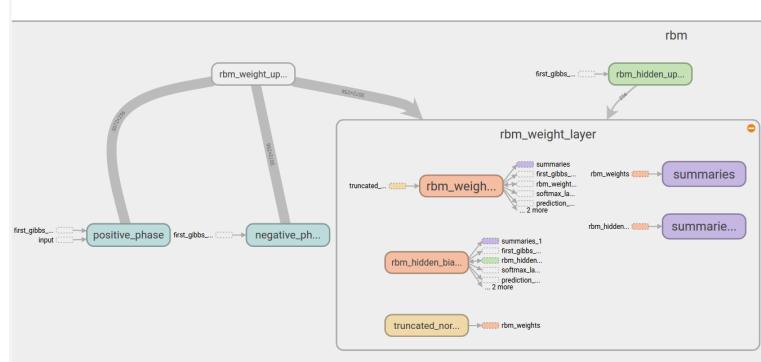
**A tanítás** A tanítást kétféle képpen végeztem el, egyrérszről definiáltam a szabad energia függvényt és a keretrendszerrel számoltattam ki a (2.6) függvényből és különböző beépített optimalizátorokkal teszteltem, gradientdescenttel és adaptív gradiens (ADAM) optimizátorral. Másrészt közvetlen kiszámoltam a gibbs sampling utáni értékekből (2.7), (2.8) és (2.9) függvényeket és egyszerűen csak hozzáadtam őket a súlyvektorhoz. A ???. ábra a hálózat madártávlati struktúráját szemlélteti. Illetve a kísérletezés egy másik dimenziója az volt hogy egyes esetekben megengedtem az RBM felé kapcsolt regresszornak hogy az előre megtanult súlyokat változtassa (ezt hívjuk fine-tuning-nak), más esetben pedig nem. Könnyen kivehető hogy egy softmax réteg is hozzá van csatolva az RBM magjához, miután felügyelet nélkül tanítom az RBM-et az a réteg végzi az osztályzást. A 3.2. ábra pedig az RBM belső strukturját mutatja. Itt látszik igazán hogy a Tensorboard grafikonjai milyen kifonomult vizualizációt tesznek lehetővé. A tanítás optimalizálását a [16]-ben leírtak alapján végeztem, de az osztályozásban meg hoztak érzékelhető javulást, ezért ezeket a mérések nél nem fejtem ki részletesen.



**3.1. ábra**

### 3.4.5. A DBN struktúra

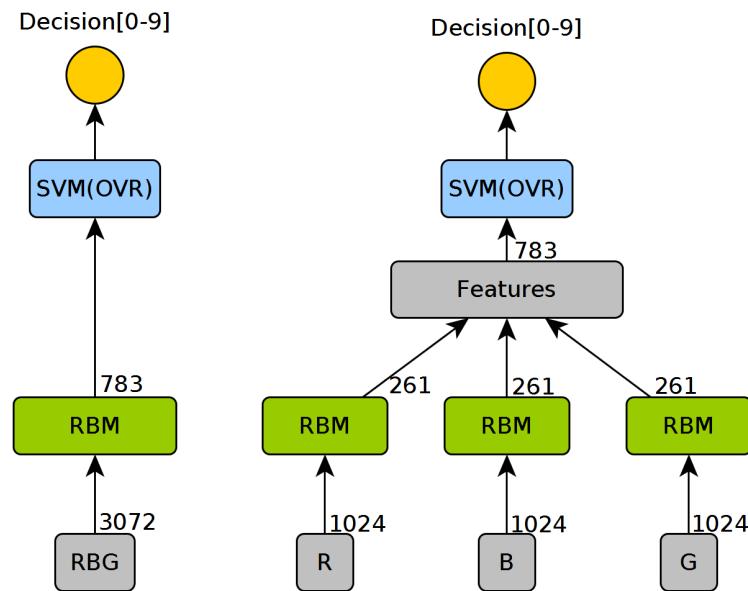
Miután az RBM struktúrát megalkottam természetesen le szerettem volna tesztelni többrétegű előre tanított hálózatok teljesítményét is. Erre nem a saját RBM hálózatomat használtam, hanem egy különböző könyvtárat, ami viszont hibás volt úgyhogy helyenként meg kellett foltoznom. Itt jött jól az a tudás amit az RBM programozásánál szereztem, mert magabiztosan mozogtam a pythonban írt tensorflow kódban.



3.2. ábra

### 3.4.6. Hibrid modellek

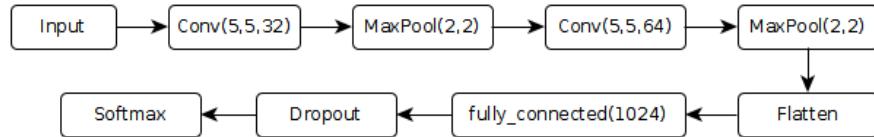
A kísérletezésem során elkezdtünk olyan architektúrákkal foglalkozni ahol a kimenet nem a neurális hálózat része, mint például az RBM osztályozók általam tesztelt variánsánál, hanem a kimeneti aktivációk valószínűségi eloszlását adtam be mintánként egy SVM-nek. Így hatalmas dimenzió csökkenést értünk el, pl 784-ről 64-re az MNIST esetében, ami jelentősen meggysorsította az SVM tanítását, anélkül hogy a nyers pixeladatokon tanított SVM-hez képest romlott volna a modell predikciós képessége. Az általam használt teszt gépen a CIFAR-10es adathalmaz nyers pixeljeit iszonyatosan hosszú idő alatt tudtam megtanítani egy SVM-nek. Ezért letömörítettem az az 3072 dimenziós CIFAR-10 adathalmazt egy 783 dimenziós térbe RBM-ek segítségével ami mindenkor jellemző kiemelést is végzett, és ott tanítottam rajta a szupport vektor gépet, remélve hogy jó osztályozási eredményeket kapok. Ezt két megközelítésben próbáltam meg, az egyik az volt amikor egyetlen RBM-em volt, 3072 bemeneti és 783 kimeneti neuronnal, a másik az volt amikor 3 RBM-et tanítottam meg, minden színcsatornára egyet, és utána ezeket egy tömbbe összefűzve adtam át az SVM-nek. A két megközelítést a 3.3. ábra szemlélteti.



3.3. ábra

### 3.4.7. A Konvoluciós modell

Több konvoluciós modell-t kipróbáltam, a méréseim egyik fő iránya az volt hogy ugyanazt a modellt probálom ki az MNIST és a CIFAR-10 adathalmazon, és megnézni hogy melyik modell hogyan reagál az adat megnövekedett komplexitására. Több hálózatot kiróbáltam, az alapmodell felépítését a 3.4. ábra mutatja. A kísérleteim arra irányultak hogy plusz teljesen összekötött rétegek hozzáadása, esetleg a konvoluciós rétegek más elrendezése milyen eredményre juttat.



3.4. ábra

Az egyes operációkat szeretném megmagyarázni röviden amiket a konvoluciós modelljeimben használtam:

- *conv(magasság, szélesség, mélység)*: Kétdimenziós, diszkrét konvolúció. Ahol a magasság és a szélesség adják meg a képfolt nagyságát. A mélység pedig hogy hány neuron legyen az adott rétegen. A kétdimenziós konvolúció definícióját a (3.4)-képlet mutatja be.
- *maxpool(magasság, szélesség)* Egy alulmintavételezési operátor a térbeli dimenziókban (magasság, szélesség). Ezzel csökkentjük drasztikusan a jellemzőterek nagyságát, és adunk transzlációs invariánciát a rendszerhez, mindenkor a legerősebb jelet viszi keresztül a pooling foltból.
- *fc(neuronok száma)*: A teljesen kapcsolt rétegeket a hálózat végére kapcsoljuk. Amíg a konvoluciós rétegek egy magas absztraktiós szinten rendelkeznek, viszonylag transzláció invariáns jellemzőteret nyújtanak a képről, addig a hálózat végén lévő teljesen kapcsolt rétegek teszik lehetővé a nemlineáris leképzéseket ebben a jellemzőtéren. Az itteni teljesen kapcsolt réteget és a hozzá tartozó nemlineáris operátorokat is az (3.1), (3.3), (3.2) írja le.
- *lrn*: Elvileg a reLu aktivációs függvény nem érzékeny annyira az adatok normalizálására, mégis kiderül a [6] publikációból hogy egy lokális normalizálási eljárás statisztikailag szignifikáns javulást képes hozni a hálózat osztályozó képességében. Ezt a matematikai struktúrát a (?) képlet szemlélteti. Ez is egy biológiai inspirált eljárás. A bilógiai neve ennek az eljárásnak a laterális inhibíció.

$$h_{ij}^k = \text{activation}((W^k * x)_{ij} + b_k) \quad \text{ahol} \quad * \quad \text{konvoluciós operátor.} \quad (3.4)$$

$$b_{x,y}^i = a_{x,y}^i / \left( k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a_{x,y}^j)^2 \right)^\beta \quad (3.5)$$

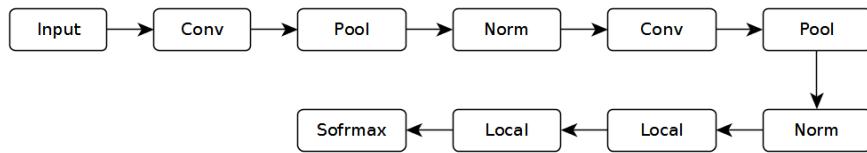
Ahol  $a_{x,y}^i$  az i-edik kernel-t alkalmazzuk az (x,y) beli pozícióra.

$b_{x,y}^i$  a normalizáció utáni eredmény

És n a vizsgált pozíció szomszédos kernel térképei, N az összes kernel a rétegen és  $\alpha, \beta$  és k pedig további hiperparaméterek.

### 3.4.8. A konvoluciós modell skálázása

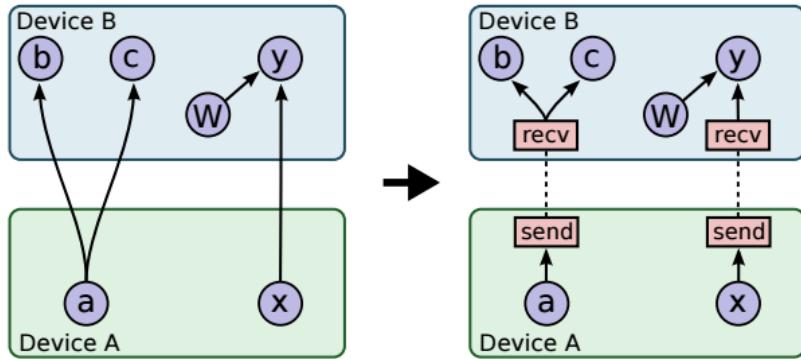
**Indoklás** Az előbbiekben bemutatott konvoluciós modell, és amikkel általánosságban a szakdolgozat keretén belül kísérleteztem könnyedén taníthatóak pár perc/óra alatt egy korszerű CPU-n. Viszont ha az ember nagyobb modelleket szeretne tanítani akkor ez már nem egy reális alternatíva. Ezekre az esetekre egy nagyobb hálózattal kísérleteztem a CIFAR-10 adathalmazon amit az alábbi instrukciók alapján készítettem el[3], aminek a felépítését a 3.5. ábra mutatja.



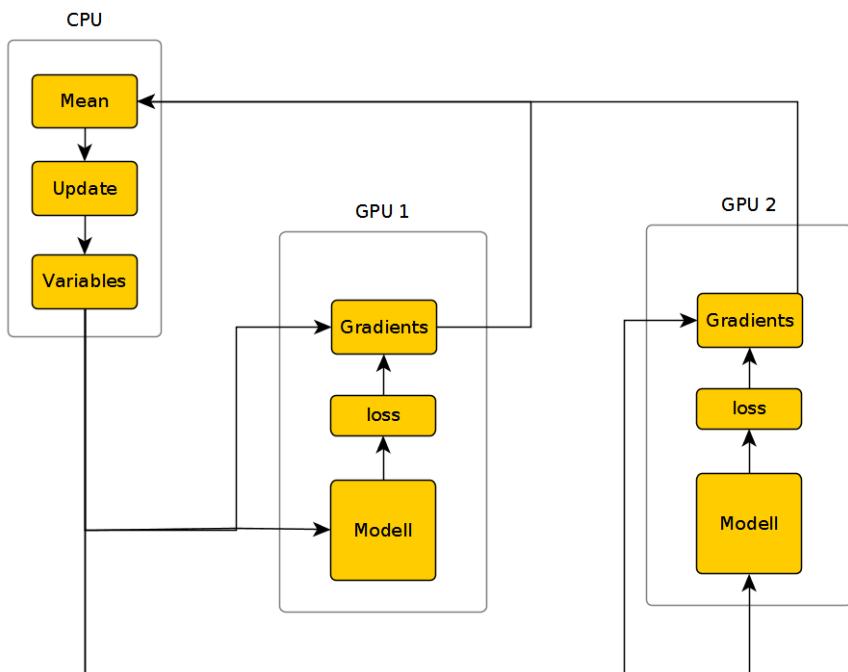
3.5. ábra

**Számítás elosztás** Ezt a hálózatot teszteltem CPU-n, 1 GPU-n, majd 2 GPU-n. A Tensorflow képes a számításokat automatikusan elosztani az eszközök között olyan módon hogy heurisztikusan megkeresi hogy mely operációk mentén érdemes szétvágni a számítási gráfot, majd azokhoz az élekhez berak küldő és fogadó csomópontokat, ahogyan a 3.6. ábra szemlélteti. Ez sok esetben teljesen megfelel az elvárásainknak, ha nem szeretnénk sokat vesződni a hálózat elosztásával, vagy amúgy is túl nagy a hálónk, és nem férne el gradiens számítással együtt rendesen egyetlen eszközön. De ha kissebb a háló és több eszközünk van akkor felmerülhet a lehetőség hogy esetleg jobban megérné a gráfot egy az egyben lemásolni az egyes eszközökre, és a súlyokat a fő memóriában tartani, majd az egyes párhuzamos futások után itt szinkronizálni a lefutásokat. Én ezt a megközelítést teszteltem le, aminek a neve torony párhuzamos tanítás, ezt a 3.7. ábra mutatja.

**Adatok dinamikus felolvasása** Természetesen az a kérdés is felmerülhet az emberben hogy ez nagyon jó hogy így el tudjuk osztani a számításokat, de az is probléma lehet ha a tanuló adatok nem férnek be a memóriába, akkor nem tudjuk rendesen tanítani őket, kivéve ha valamilyen bonyolult felolvasási kódot irunk hozzá. Hál istennek ezt nem kell megirnunk, hanem be van építve a keretrendszerbe, a Tensorflow erre direkt felolvasási pipelineokat állít a programozó rendelkezésére ennek a felépítését szemlélteti a 3.8. ábra. Egy ilyen pipelinenak a következő paramétereit vannak:



3.6. ábra



3.7. ábra

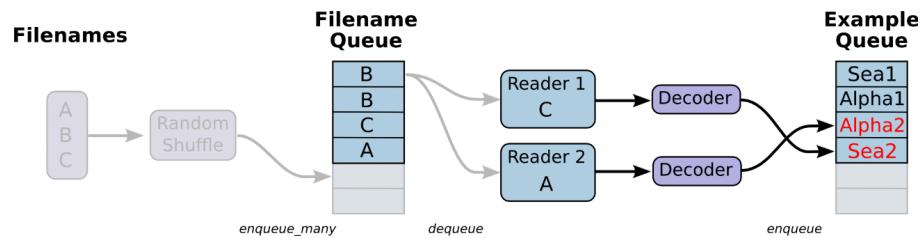
- Bemeneti file nevek.
- A felolvasásra és előfeldolgozásra használni kívánt szálak száma.
- A megállási kritérium
- A pipeline mérete, hogy hány még meg nem tanított kép legyen mindig a pipelineban.

Ezek után a keretrendszer gondoskodik arrol hogy a szálak minden stratifikáltan, külön filekből olvassanak fel, illetve ha példányosítunk egy koordinátor objektumot és berrgisztráljuk hozzá a pipeline-t akkor a szálak közötti hibakezelésről is gondoskodik, hogy egy szál hibája esetén ne akadjon ki a rendszer. Ezen felül definiálhatunk még előfeldolgozási lépéseket minden képhez, az általam használt implementációban például egyszerre 20 feldolgozatlan

képet tart a memóriában, ezeket először közelítőleg fehéríti (ez is beépített funkció), majd ezt az adathalmazt négysszerezi a következő képpen:

- A képek véletlenszerű tükrözése.
- A kép világosságába való véletlenszerű zaj bevezetése.
- A kép kontrasztjának véletlenszerű torzítása.

Ez jelentősen növeli a háló általánosító képességét. Majd ezeket az adatokat beadja egy új sorba, ami véletlenszerű batcheket generál a 16 szál képeiből, és megadhatjuk hogy hány modell vegye kis és dolgozza fel őket egyszerre. Esetben a torony-parallel elrendezésben ez a GPU-k számától függ.



3.8. ábra

## 4. fejezet

# A mérési eredményeim összegzése

Ebben a fejezetben szeretném bemutatni az általam elkészített hálózatok futásának mérési eredményeit. És ezeknek aspektusait:

1. A keretrendszer általános teljesítménye.
2. A baseline eredmények bemutatása.
3. Az MLP hálózattal elért eredmények és az ezekből levont tanulságok.
4. A konvoluciós hálózatokkal elért eredmények és ennek összegzése.
5. A hibrid hálózatokkal elért eredmények.

Illetve nagyon sok olyan dolog is van amiről tudom hogy hozzáartozik a modellalkotói munkához, de itt nem foglalkoztam vele, egyrészt a véges erőforrások, másrészt pedig a véges idő miatt. Illetve nem láttam volna hogy hozzáadtak volna jelentős mértékben a szakdolgozatom céljához. Ezek voltak:

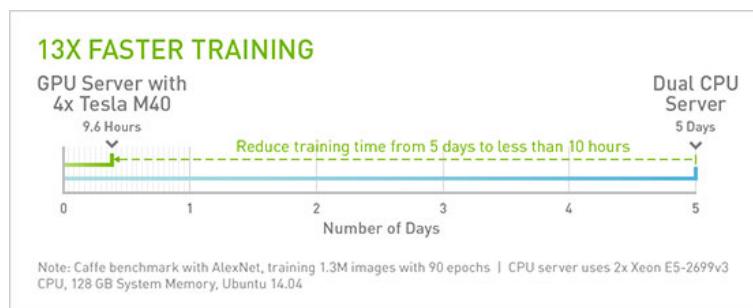
- A hiperparaméterek végletekig menő tuningolása grid search-el.
- A hiperparaméter halmazok összehasonlítása, akár kereszt validációval, és egy végeleges hiperparaméter beállítás kiválasztása a teszt halmazon való kiértékeléshez.
- A lépésszámok pontos kíkísérletezése, és korai leállási feltételek megfogalmazása és implementálása.
- Az adatok átfogó előfeldolgozása. minden adatot normalizáltam, de nem végeztem több előfeldolgozást, pl fehérítést.
- Az adathalmaz bővítése. Ide tartoznak a képek véletlenszerű tükrözése, kivágása, rotációi, egyes értékek disztorciói (pl kontraszt).

Tehát a modelljeimet empirikus paraméterekkel, a nyers adatokon teszteltem. Mindeközben tisztában voltam az összes architektúra elméleti teljesítményének a határával, ez is egy érdekes tanulság volt, hogy ilyen "naiv" megközelítéssel milyen közel tudok jutni egy-egy kutatás eredményéhez.

A gép specifikáció amin általánosságban teszteltem (eltekintve a GPU skálázás résztől) a következők:

- Memória: 8 Gigabyte DDR4
- CPU: Intel Core i7-4702MQ CPU @ 2.20GHz x 8

A gépben elhelyezkedő grafikus kártyát nem használtam, az Nvidia hibás eszköz meghajtó programja miatt. Egy GPU-n mért teljesítmény karakterisztikák eltérhetnek, főleg egy olyan hatékony implementáció mellett mint a cuDNN, az Nvidia által fejlesztett mély neurális rendszerekhez használt könyvtár, ami nagyon hatékony implementációkat tartalmaz a Neurális hálózatok legtöbb operációjára. A mértékek szemléltetéséhez a 4.1. ábra szemlélteti. Érdemes hozzáenni hogy egyetlen Tesla 40M ára most kb 1,3 millió forint. Amíg a CPU-s tesztben két Intel Xeon E5-2699v3 szerepelt, jelenlegi darab ára hozzávetőlegesen 1 millió forint. Tehát a GPU-s megoldásnál hozzávetőleges 2.6-szor drágább volt csak a megoldás magját képező számítási felszerelés. Nyílván ilyen eszközököt már csak nagy tőkével rendelkező szervezetek engedhetnek meg maguknak.



**4.1. ábra.** Az alexNet tanítása high end szerver két CPU-s intel xeon e5-2699v3 szerveren, és 4 elemből álló Tesla M40 griden. Forrás:<http://images.nvidia.com/content/products/hpc/faster-performance-than-cpu.png>.

## 4.1. A keretrendszer általános teljesítménye

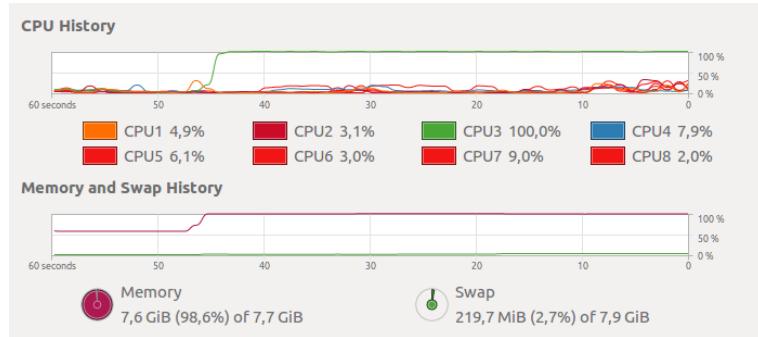
Ebben a részben azt vizsgáltam hogy a tensorflow hogyan bánik az erőforrásokkal. Ennek érdekében több dolgot tettem:

- Egyszerű rendszerszintű méréseket futtattam.
- A Tensorboardon keresztül vizsgáltam a hálók teljesítmény és erőforrás karakterisztikáját.
- Egy konvoluciós hálót kiskáláztam a CPU-ról 1 GPU-ra, majd 2 GPU-ra.

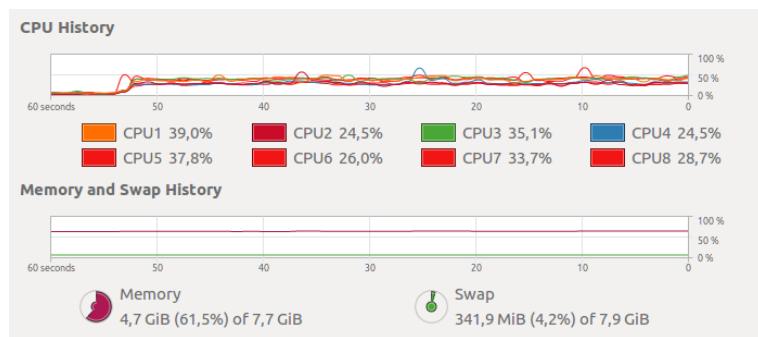
## 4.2. Rendszer szintű mérések.

Amikor nem sikerült lefuttatnom egy logisztikus regressziót Scikit segítségével a CIFAR-10 adathalmazon mert a teszt gép használhatlanná vált, akkor felüttötte a fejét a kérdés nálam hogy mégis hogyan viszonyul teljesítményben a tensorflow a Python de facto gépi tanulás

eszközéhez, a Scikit-learn könyvtárhoz. A két mérés erőforrás karakterisztikáját a 4.2. ábra és a 4.3. ábra mutatja, ahol az előbbi a Scikit-learn-ben elindított logisztikus regresszió, az utóbbi pedig a Tensorflowban megvalósított közelítő Logisztikus Regresszió.



4.2. ábra. A Scikit-learnben idított logisztikus regresszió erőforrás igénye.



4.3. ábra. A Tensorflowban idított logisztikus regresszió erőforrás igénye.

Mint látható a scikit-learnben elindított futtatás a teszt gép összes memóriáját felemészítette, illetve a magok kihasználása is nagyon rossz volt, mindenki egyetlen magot használt ki. Ezzel szemben a Tensorflowban approximált modell nagyon egyenletes magkihasználtság mellett minimális memória igénytől oldotta meg a feladatot. Természetesen nem állítom hogy a két feladat identikus volt, mivel más algoritmusokkal jutottak el a végeredményig. További kutatásaimból kiderült hogy ez a Scikit-learn alapértelmezett optimalizációs algoritmusának köszönhető amit a liblinear könyvtár valósít meg. Amikor ezt kicseréltem SAG megoldóra, akkor a gép lefagyásának problémája eltünt, de még mindig konvergencia problémák léptek fel. Az SAG egy kifejezetten új fejlemény a numerikus optimalizáció területén, amit egy 2013-as publikáció mutat be "Minimizing Finite Sums with Stochastic Average Gradient"[20]. Ezen felül még megprobáltam az adathalmazt a newton konjugált-gradiens módszerrel megtanulni, de ez is sikertelen lett az erőforrások hiánya miatt. Ami érthető, hiszen a Newton-cg optimizátor gyors konvergenciát ígér, de még a sima liblinear megoldónál is nagyobb erőforrás igényel.

Ez a mérési sorozat úgy gondolom hogy egyértelműen rávilágít a numerikus optimalizáció kiemelkedő fontosságára a gépi tanulás applikációkban. Az adathalmaz amin teszteltem a metódusokat nem volt nagy, a CIFAR-10 mindenki 180 megabyte. Amennyiben ezeket a méréseket az akadémia szerverein végezem volna el, akkor gond nélkül taníthattam volna akármelyik módszerrel. Viszont így, hogy a tesztrendszer is igen korlátozott jól

megvilágította a nagy adathalmazokon való tanulás egyik nagy problémáját. Habár a mostani szereink már hatalmas erőforrásokkal bírnak, a klasszikus módszereink amik folyamatosan az egész adathalmazzal való interakciót igénylik, mint mondjuk az eredeti logisztikus regresszió vagy svm tanító algoritmusok mégsem lesznek használhatóak egyszerűen az adathalmaz pusztá nagysága miatt. Ezért is fontos hogy a mostani kutatások nagyrésze a numerikus optimalizáció terén a sztochatsztikus módszerekre koncentrál amik véletlenszerűen összeválogatott kis tanító halmazokkal approximálják a teljes adathalmaz tulajdonságait.

//TODO: GPU

### 4.3. A baseline eredmények ismertetése

Mint már említettem, a két alap metódus amihez a saját eredményemet mértem a logisztikus regresszió, és egy egyszerű SVM volt, amihez polynomiális kernelt használtam 3-as fokszámmal és az első rendű tagokat használva. Az eredményemet az eredményiemet az MNIST adathalmazon a 4.1. táblázat, és a 4.2. táblázat szemlélteti.

**4.1. táblázat.** A baseline mérések eredménye az MNIST adathalmaz nyers pixeljein, használt könyvtár: Scikit-learn

tanuló metódus	Teszt halma hiba százaléka
Logisztikus regresszió	8%
SVM poly kernel fok: 3 coef0: 1	6%

Az MNIST-en elért eredmények nem a legjobbak, de egészen optimális eredmények ilyen egyszerű eljárásokkal is. Ezekben sokat lehet javítani a képek előfeldolgozásával, de ez egy külön szakdolgozat témája lehetne, úgyhogy ezzel most itt nem foglalkozok. minden struktúrát a nyers adatokon kezdtem el tréningezni, klasszikus előfeldolgozási lépések alkalmazása nélkül. A teljesség kedvéért megemlítem hogy a legjobb eredmény amit SVM-el értek el az MNIST adathalmazon annak a felépítése a következő volt: *Virtual SVM, deg-9 poly, 2-pixel jittered*, előfeldolgozási lépésként csak kiegyenesítést használt (deskewing), az elért teszt hibaszázalék pedig 0.56% volt.

**4.2. táblázat.** A baseline mérések eredménye az CIFAR-10 adathalmaz nyers pixeljein, használt könyvtár: Scikit-learn

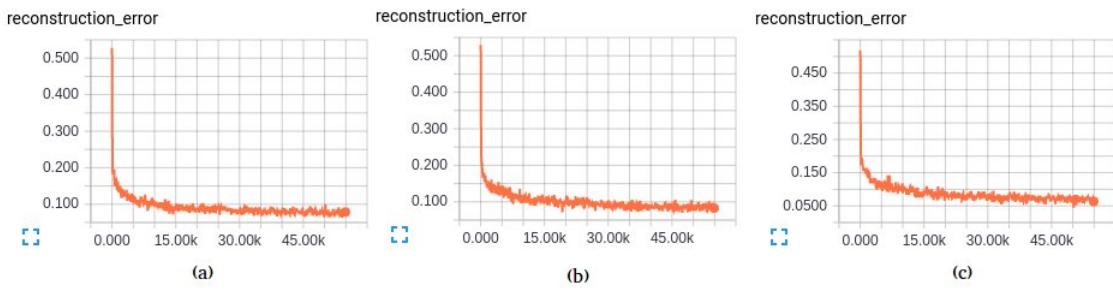
tanuló metódus	Teszt halma hiba százaléka
Logisztikus regresszió	62%
SVM poly kernel, coef0=1 fokszám=3	55%

Mint látni lehet itt ezek a metódusok már közel sem teljesítenek annyira fényesen. A legjobb eredmény amit elértek az adathalmazon SVM-el az 25,5% volt. Ez egy ICML 2010-es konferencián bemutatott eredmény volt, amit a "Improved Local Coordinate Coding using Local Tangents"[19]-ban publikáltak.

### 4.4. Az RBM struktúra optimalizálása

Ahogyan azt a saját munkám leírásánál is taglaltam több féle képpen végeztem el méréseket az RBM struktúrával hogy megfigyeljem hogyan hasonlunak egymáshoz. Először a szabad

energia függvények külöbségét definiáltam a (2.6) szerint, és ezt optimalizáltam, majd simán hozzáadtam az általam lederivált gradienseket. A rekonstrukciós függvények lefutását a 4.4.-ábra szemlélteti. Látható hogy a 3 megközelítésnek a lefutásában és erőforrás felhasználásában is jelentős különbség van, ezt a 4.3. táblázat szemlélteti. Jól látszik hogy az adaptív optimizátor megnövekedett számításigényel jár, az SGD ugyanannyi memóriát igényel, mert kb ugyanazt az implicit konstruált gráfot használja de kevésbé terheli a processzort, amíg a kézi optimalizálás még ennél is lényegesen kevesebb erőforrást igényel minden tekintetben. Éppen ezért a kézzel kiegészített megoldást választottam. Ez a megoldás jobban is skálázódik több számítási egységre, ha nagyobb RBM-et akar tanítani az ember, mert a gradiens frissítő gráf miatt nem fog kommunikációs overhead fellépni az eszközök között, mivel ez a gráf nem jön létre.



**4.4. ábra.** Az RBM rekonstrukciós hibájának csökkenése a tanító batcheken.  
Az (a) ADAM optimizátorral, (b) SGD optimizátorral, (c) a gradiensek manuális megadásával.

**4.3. táblázat.** Az RBM erőforrás mefelhasználása és hiba százaléka 10 epoch után, 256 neuron mellett különöző optimizátorokkal.

optimizátor	Batch idő	Memória	Rekonstrukciós hiba
ADAM	11.8 ms	10.7 MB	0.077
SGD, tanulási ráta: 0.1	7.78 ms	10.7 MB	0.087
SGD, tanulási ráta: 0.01	7.78 ms	10.7 MB	0.084
SGD, tanulási ráta: 0.001	7.78 ms	10.7 MB	0.12
Manuális	2.67 ms	5.62 MB	0.07

## 4.5. Az MLP-vel elvégzett méréseim eredményei

### 4.5.1. Az MNIST mérések

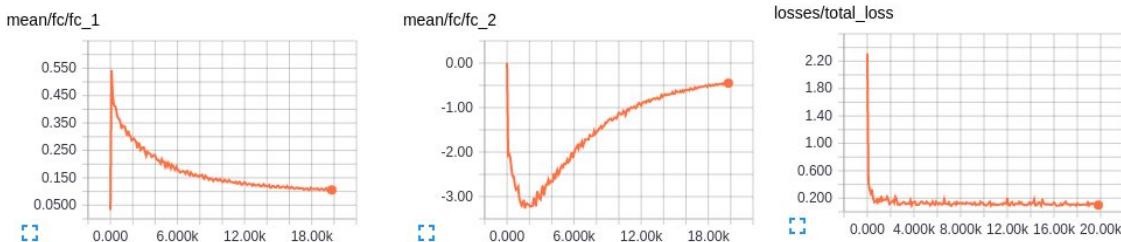
A többrétegű perceptronokról ismeretes hogy a rétegek között teljes kapcsolat van, és az adathalmaznak semmilyen tulajdonságát nem építik bele az architektúrába a priori. A konvoluciós hálózatok esetében, viszont a transzlációs invariancia előre feltételezett, és maga a hálózat tartalmazza a kényszereket. Ez alapvetően abban is megnyílvánul hogy a hálózat nem egy három dimenziós bemenetre (magasság, szélesség, mélység(RGB)) támászaszkodik, hanem egy egyszerű vektorra. Az MNIST-en elért eredményemet egy ilyen egyszerű struktúrával a 4.4. táblázat mutatja be, a méréseket a fejezet elején ismertetett tesztgép CPU-ján végeztem. A bemenet egy 784 elemű vektor volt, ami praktikusan az MNIST adathalmaz egyetlen vektorba lapítva sorról sorra.

**4.4. táblázat.** A saját MLP teljesítménye az MNIST adathalmazon, 20 000 tanító batch után. Bemenetek száma: 784, reLu aktivációval, L2 reguralizációval, aminek az együtthatója 0.0005 volt. A súlyokat csonkolt normál eloszlással inicializáltam, standard szórás: 0.1, középérték: 0.0.

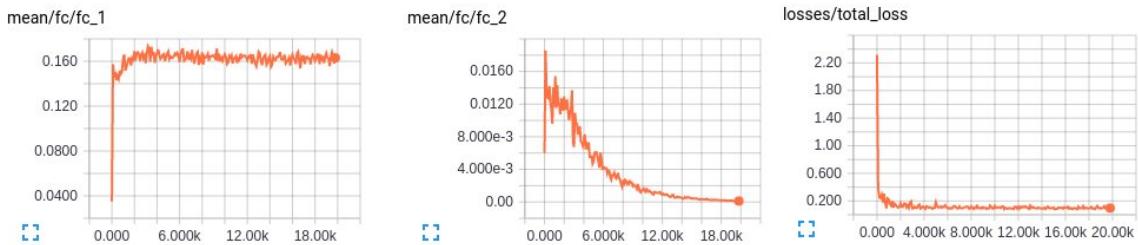
neuron struktúra	Optimalizáló algoritmus	Teszt szet hiba százalék
10	SGD learning-rate: 0.5	8%
1024-10	SGD learning-rate: 0.5	3.9%
400-10	SGD learning-rate: 0.5	2.02%
400-10	Adam	2.04%
800-10	Adam	2.21%
800-10	SGD learning-rate: 0.5	2.00%

Látható hogy az első struktúra a logisztikus regresszióval azonos eredményt ad, ami nem meglepő, hiszen egy egyrétegű struktúra softmax függvényel a végén praktikusan ugyanakkora általánosító képességgel rendelkezik mint egy klasszikus módszerekkel tanított multinomiális logisztikus regresszor. Ezen az eredményen lényegesen tudtam javítani miután hozzáadtam egy 1024 elemű rejtett réteget, ami növelte a háló általánosító képessegét, de 20'000 lépés alatt nem volt képes túltanulás nélkül megtanulni a feladatot. Ahogyan csökkentettem a hálózat méretét az csökkentette a túltanulás mértékét és jelentős teljesítmény növekedést hozott. A 2.00%-os hiba eredmény már elmondható hogy egészen közel van az eddigi legjobb elérte eredményhez 1 rejtett réteggel, ami 0.7%. Azt vártam volna hogy az Adaptív momentum közelítő (adam) módszer javít a hálózat eredményén, de nem így lett. Általában az Adam optimalizátor előnye abban rejlik hogy mivel a momentumot és a tanulási rátát adaptívan számítja minden batchre, ezért kevesebb hiperparamétert kell állítani. Ez természetesen megkövetelte számítás igénytől szokott járni.

A 4.5. ábra és a 4.6. ábra mutatja hogy mekkora hatással van az gradiens keresési eljárás a megtanult súlyokra. Látszik hogy a két eredmény merőben más optimumra állt be. Mint ahogyan a 4.5. tábláztból is látszik, az SGD-t használó hálózat sokkal közelebb került a teszt adathalmaz általános képéhez. Az is nagyon jól látszik a képeken hogy az ADAM optimalizátor úgy alakította az együtthatókat, hogy lényegesen kisobb változtatásokat tegyen a felületen, ezért egyáltalán nem ugrált annyira a súlyok értéke mint az SGD-nél. Viszont mind a kettő a veszteségfüggvény stabilítási pontját olyan 2000 iteráció után érte el, onnantól már csak oszcilláltak a maradék 18000 lépésben.



**4.5. ábra.** A veszteségfüggvény és a hálózat súly átlagainak a rétegenkénti alakulása ADAM optimizátor mellett.



**4.6. ábra.** A veszteségfüggvény és a hálózat súly átlagainak a rétegenkénti alakulása SGD optimizátor mellett.

#### 4.5.2. A CIFAR-10 mérések

**Az MNIST és a CIFAR különbségei** Miután az MNIST adathalmazon sikeres méréseket végeztem egészen egyszerű MLP hálózatokkal kiváncsi lettem hogy ezek a hálózatok mennyire viszik át a teljesítményüket egy fokkal bonyolultabb adathalmazra. A két adathalmaz tulajdonságai egymáshoz képest:

- Mindkét adathalmaz 10 osztályt tartalmaz.
- Mindkét adathalmaz 10'000 teszt képet tartalmaz, 1000-et osztályonként.
- A CIFAR-10-es adathalmaz 10'000-rel kevesebb tanító képet tartalmaz, ez osztályonként 1000 darab.
- Amíg az MNIST 28x28-as fekete fehér képeket tartalmazott, addig a CIFAR-10 32x32-es színes képeket tartalmaz.

Ami rögtön szembetűnik hogy amíg az MNIST 784 dimenziót tartalmazott, addig a CIFAR-10 egy 3072 dimenziós adathalmaz, tehát az adathalmaz komplexitása ha csak tisztán a dimenziókat nézzük akkor a négyzetesére nőtt. Ennek megfelelően a hálózatok ha ugyanazokat a hálózatokat próbáljuk használni, akkor drasztikus teljesítmény csökkenést vagyunk kényetlenek tapasztalni.

**4.5. táblázat.** A saját MLP teljesítménye az CIFAR adathalmazon, 20 000 tanító batch után. Bemenetek száma: 3072, reLu aktivációval, L2 reguralizációval, aminek az együtthatója 0.0005 volt. A súlyokat csonkolt normál eloszlással inicializáltam, standard szórás: 0.1, középrték: 0.0.

neuron struktúra	Optimalizáló algoritmus	Teszt szet hiba százalék
10	SGD learning-rate: 0.5	80%
10	SGD learning-rate: 0.001	75%
400 - 10	SGD learning-rate: 0.01	79 %
1024 - 10	SGD learning-rate: 0.01	90%

#### 4.5.3. Az előtanítás eredményeinek összefoglalása

Arra a számonra meglepő eredményre jutottam hogy az előtanítás az az én méréseim közben minden esetben csak rontott a hálózat klasszifikációs teljesítményén. Az általam várt eredmény az lett volna, hogy az előtanítás jelentősen javítani fog a hálók teljesítményén, főleg nagy neuron számnál. A 4.6. táblázat és a 4.7. táblázat mutatja be az eredményeimet.

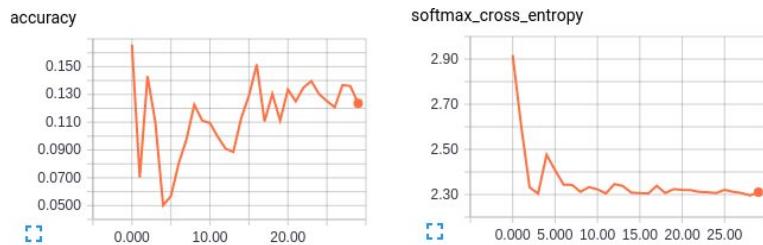
**4.6. táblázat.** Az előtanított MLP hálózatok eredménye a MNIST adathalmazon, 25 elemű batchekkel.

neuron struktúra	nem-felügyelt/felügyelt epoch szám	Optimalizáló algoritmus	Teszt szet hiba százalék előtanítássá / nélküle
800 - 10	15/30	SGD learning-rate: 0.05	92% / 0.9%
500-500-2000-30	15/30	learning-rate: 0.001	88.5% / 3.2%

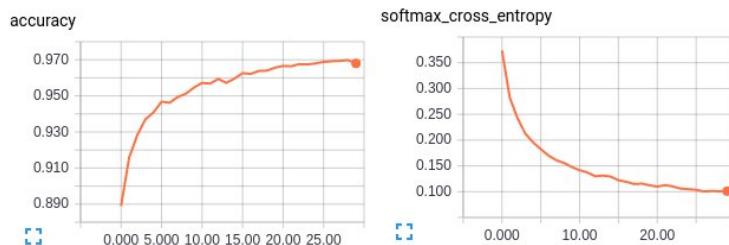
**4.7. táblázat.** Az előtanított MLP hálózatok eredménye a CIFAR adathalmazon, 25 elemű batchekkel.

neuron struktúra	nem-felügyelt/felügyelt epoch szám	Optimalizáló algoritmus	Teszt szet hiba százalék előtanítássá / nélküle
400 - 10	10/10	SGD learning-rate: 0.001	91% / 55%
500-500-2000-30	10/10	learning-rate: 0.001	88.5% / 57.5%

Úgy magyarázom ezeket az elszomorító eredményeket hogy az előtanítás következtében egy rossz lokális minimumba ragadt be a háló, ahonnan nem tudott kijönni egyik esetben sem. Ahhoz hogy ezt a feltevésemet igazolja mmegnéztem az mnist adathalmazon tanított nagyméretű (500-500-2000-30 neuron) hálónak a veszteségfüggvényét és a pontosságának a növekedését a validációs adathalmazon. Ez előtanítás nélküli háló mért eredményei a 4.7. ábrán, az előtanítással tanított hálónaé pedig a 4.8. ábrán láthatóak. Az ábrákon szépen kijön hogy amíg az előtanítás nélküli háló mindenkorábban konvergált, addig az előtanított háló beragadt egy rossz lokális minimumba, ahonnan nem tud kikerülni. Ebből azt szűrtem le, hogy érdekes módon nem feltétlen azok a jó jellemzők osztályozáshoz, amik a rekonstrukcióhoz. Pedig nekem egészen intutív lett volna, és az irodalomban is sokat olvastam róla hogy ez az előnye az RBM-ekkel való előtanításnak.



**4.7. ábra.** A 500-500-2000-30 neuronú előtanított MLP veszteségfüggvényének, és a validációs halmazon való pontosságának alakulása.



**4.8. ábra.** A 500-500-2000-30 neuronú nem előtanított MLP veszteségfüggvényének, és a validációs halmazon való pontosságának alakulása.

## 4.6. A konvoluciós hálózatokkal elvégzett méréseim eredményei

Miután láthattuk hogy az MLP-k már nehezen bírkóznak meg a CIFAR-10 komplexitású feladatokkal a figyelmemet az elméleti részben oly sok helyet elfoglaló konvoluciós architektúrák felé fordultam. Mivel ezek az architektúrák érik el jelenleg a legjobb eredményeket a neurális jelfeldolgozás széles területén, nem csak képosztályozásban hanem akár beszédszintetizálásban is, ezért nagy reményekkel fordultam ezekhez a struktúrákhöz. Az eredményeimet a 4.8. táblázat mutatja az MNIST adathalmazon, és a 4.9. a CIFAR-10 adathalmazon. Az alábbi felsorolás az általam használt hálózatokat írja le.

1. conv2d(5,5,32) -> flatten -> softmax
2. conv2d(5,5,32) -> maxpool(2,2) -> flatten -> fullyconnected -> dropout\_0.5 -> softmax
3. conv2d(5,5,32) -> maxpool(2,2) -> conv2d(5,5,64) -> maxpool(2,2) -> flatten -> fullyconnected(1024) -> dropout(0.5) -> fullyconnected(1024) -> softmax
4. conv2d(5,5,32) -> maxpool(2,2) -> lrn -> conv2d(5,5,64) -> lrn -> maxpool(2,2) -> flatten -> fullyconnected(1024) -> dropout(0.5) -> fullyconnected(1024) -> softmax

**4.8. táblázat.** A saját CNN teljesítménye az MNIST adathalmazon, 20 000 tanító batch után. Bemenetek száma: 3072, reLu aktivációval, L2 reguralizációval, aminek az együtthatója 0.0005 volt. A súlyokat csonkolt normál eloszlással inicializáltam, standard szórás: 0.1, középérték: 0.0.

A struktúra sorszáma	Optimalizáló algoritmus	Teszt szet hiba százalék
1	ADAM	8%
2	ADAM	1.2%

**4.9. táblázat.** A saját CNN teljesítménye az CIFAR adathalmazon, 30 000 tanító batch után. Bemenetek száma: 3072, reLu aktivációval, L2 reguralizációval, aminek az együtthatója 0.0005 volt. A súlyokat csonkolt normál eloszlással inicializáltam, standard szórás: 0.1, középérték: 0.0.

A struktúra sorszáma	Optimalizáló algoritmus	Teszt szet hiba százalék
1	ADAM	33.23%
2	ADAM	33.51%
3	ADAM	26.9%
4	ADAM	26.0%

Látszik hogy az MNIST-en is egész jól teljesítenek a konvoluciós hálózatok, de arra az adathalmazra elég egy egyszerű MLP modellező képessége is. Ami viszont érdekesebb hogy a CIFAR10-en, ahol láttuk hogy az MLP hálózatok igen szerény eredményeket érnek el, ott a CNN hálózatok már elkezdenek az MLP hálózatoknál lényegesen jobb eredményeket produkálni. Ennek a magyarázata a transzlációs invariancia által a súlyokra vetített kényszerek, amik természetesen jelen vannak a képi adatokban. Cserébe viszont a konvolúció miatt lényegesen megnő a számítási igény, és a hálózat végén található teljesen kapcsolt rétegek miatt bekövetkezett paraméter tér robbanás miatt hosszabb ideig is kell tanítani ezeket a hálózatokat. A lokális válasz normalizálás az én esetemben is 1%-ot javított a teszt hiba százalékon, mint ahogyan azt Hintonék is tapasztalták a "ImageNet Classification

with Deep Convolutional Neural Networks"[6] nevű publikációban. Cserébe a gradiensek kiszámolásának az idejét 174 ms-ről, 355 ms-re emelték. Viszont megemelkedett memória igényük csak 3 MB volt. Ez érhető, hiszen ez az operáció nem foglal plusz memória területet mint a paraméterek, pusztán nehéz a deriváltját kiszámítani.

A Legjobb eredmény amit konvoluciós hálózattal a CIFAR adathalmazon elértem az 18% százalékos hiba volt, a fent ismertetett lokális válasz normált struktúrával, csak 2 GPU-n 300'000 lépésen keresztül tanítottam. Ez már egy egészen tisztességes eredménynek számít az adathalmazon.

#### 4.7. A konvoluciós és az MLP hálózatok erőforrás igényének az összehasonlítása

Észrevételeim szerint az MLP hálózatok kis költségráfordítás mellett teljesítenek olyan jól az MNIST adathalmazon mint a konvoluciós hálózatok. Ugyanakkor úgy vélem hogy ez az előny csak az MNIST végtelen egyszerűségéből fakad. A CIFAR10-es adathalmazon már jól észrevehető a konvoluciós hálózatok elsöprő fölénye, ha komplex képosztályozási feladatokról van szó.

Két struktúrának mértem össze a paramétereit az MNIST adathalmazon. Egy MLP-ét, és egy konvoluciós hálózatét. A hálózatok az alábbi struktúrával épültek fel, ebben a sorrendben:

- input(784) -> fc(500) -> fc(500) -> fc(2000) -> fc(30) -> fc(10)
- input -> conv2d(5,5,32) -> pool(2,2) -> conv2(5,5,64) -> pool(2,2) -> fc(1024)-> dropout(0.5) -> fc(1024) -> softmax

Érdemes megjegyezni hogy az MLP paraméter tere kb 1,8 millió paraméterből áll, amíg a konvoluciós hálózat kb 2,7 millió paraméterből. Habár eleinte úgy éreztem hogy a konvoluciós háló paraméter tere kisebb lesz, mégis az intuicióm becsapott.

Mindkét hálózatban a gradiens kiszámítása igényelte a legtöbb erőforrást. Ennek a számításnak a részleteit a 4.10. tábla mutatja az mlp hálózatra, és a 4.11. tábla a konvoluciós hálózatra. A táblázatokból jól látható hogy a kovoluciós hálózat lényegesen több erőforrást fogyasztott. Ha az erőforrások növekedése lineárisan skálázódna a paraméter tér beli növekedéssel, akkor jóval kisebb növekvény lett volna várható.

**4.10. táblázat.** Az MLP hálózat megmagasabb erőforrás igényű részei az MNIST adathalmaznál

Hálózati rész	Memória	Számítási idő
gradiens	20.3 MB	12.5 ms
Az első teljes réteg gradiense	3.15 MB	12.1 ms
A második teljes réteg gradiense	2.03 MB	9.82 ms
ADAM optimizer	8 B	9.37 ms
Harmadik teljes réteg	750 KB	3.09 ms
Negyedik teljes réteg	11.3 KB	2.82 ms

A fenti két táblázatból több érdekes következtetést is levontam, ezeket az alábbi felsorolásban összegzem:

- A konvoluciós háló paraméter tere lényegesen nagyobb lett, pedig én intuitívan kisebbnek

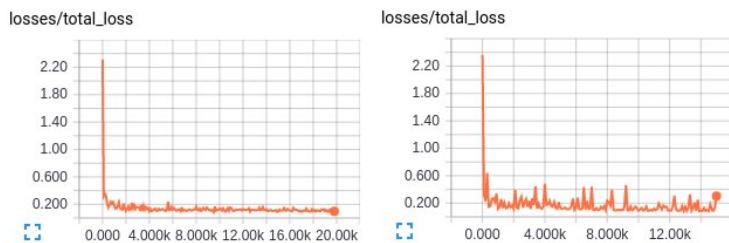
**4.11. táblázat.** A konvoluciós hálózat megmagasan erőforrás igényű részei az MNIST adathalmaznál

Hálózati rész	Memória	Számítási idő
gradiens	114 MB	94.8 ms
A második konvoluciós réteg gradiense	44.6 MB	90.9 ms
Az első konvoluciós réteg gradiense	27.5 MB	94.7 ms
ADAM optimizer	8 B	50.1 ms
Első teljes réteg	192 KB	43.6 ms
Második előrecsatolt konvoluciós ág	2.3 MB	36.8 ms

gondoltam volna mint az MLP-ét. Azt sejtettem hogy a paraméter tér robbanását a két teljesen csatolt réteg fogja okozni, de ennyire nagyra nem számítottam.

- A számítást magasan a gradiens kiszámítása dominálja mind a két hálónál, viszont ahhoz képest hogy a CNN paraméter tere kb 1.5-szer akkora, a gradiens számítás ötször annyi memóriát, és kb nyolcszor annyi számítási időt igényel mint az MLP esetében.
- A CNN-ben a paraméter tér robbanásáért felelős egy batch számításánál elhanyagolható többletmunkát jelentenek.

A 4.9. ábra azt mutatja hogy az MNIST-en legjobban teljesítő, 800 neuronos MLP, és az előbb említett CNN veszteségfüggvényei hogyan viszonyulnak egymáshoz. Látható hogy amíg a CNN egy kicsit ugrál, nem tud beállni egy stabil pontba az adathalmaz kicsi mérete miatt, addig az MLP gond nélkül hoz konzisztens eredményeket a veszteségfüggvényben.



**4.9. ábra.** A Tensorflowban idított logisztikus regresszió erőforrás igénye.

Ugyanez a két metrika a CIFAR10 adathalmazon a ??, ábrán látható. tisztán látszik hogy ide ez az MLP hálózat már kevés volt. Sőt, én sokkal mélyebb MLP függvényekkel sem tudtam lényegesen jobb eredményt elérni, ezt tisztán mutatják a 4.5. eredményei is.

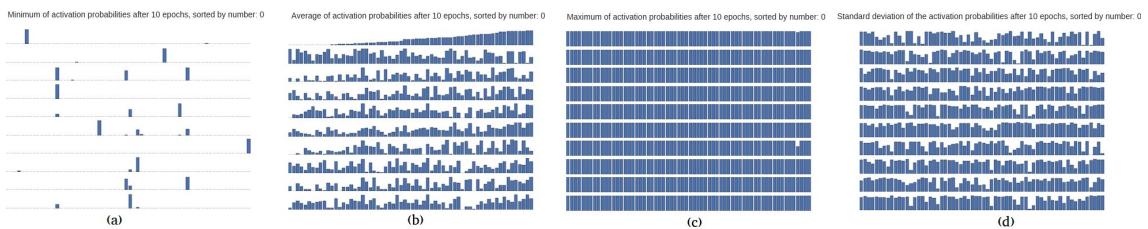
## 4.8. A hibrid architektúrák eredményei

**MNIST** A fent ismertetett hibrid architektúrákból az MNIST adathalmazzal csak azt probáltam ki, ahol 1 darab RBM van. Ezt megtettem a scikit-learn SVM-jében (ez a libsvm python kötésekkel végülis) elérhető összes kernel implementációval, és 1-2 paraméter kombinációjukkal. Az eredményeket a 4.12. táblázat szemlélteti. A táblázaton látszik hogy a legtöbb kernel nagyjából ugyanúgy teljesít, kivéve a sigmoid kernel, mert az kiemelkedően rosszul teljesít a többihez képest.

**4.12. táblázat.** Az  $RBM + SVM$  kombináció eredménye az MNIST adathalmazra különböző kernelekkel.

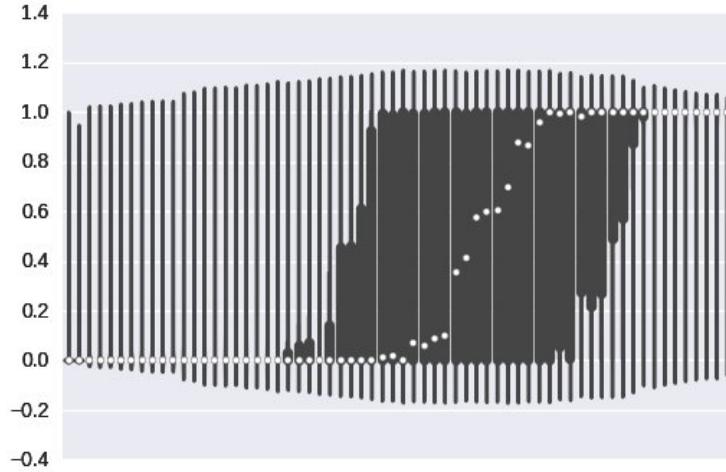
A kernel	RBM méréte	Teszt szet hiba százalék
linear	64	7.3%
rbf	64	6.0%
poly coef0=0 degree=2	64	7.1%
poly coef0=1 degree=2	64	5.8%
poly coef0=0 degree=3	64	6.0%
poly coef0=1 degree=3	64	5.6%
sigmoid coef0=0	64	89%
sigmoid coef0=1	64	89%

**Aktivációk vizsgálata** Miután ilyen jó eredményeket hozott a 64 rejtett neuronos RBM az adathalmazon elkezdtem érdeklődni hogy vajon már a neuronok aktivációiból látszani fog-e az, hogy a kép melyik osztályba fog tartozni. Ehhez azt csináltam hogy az összes képre az MNIST-ben legeneráltam a rejtett neuronok aktivációs valószínűségét, és osztályonként vettetem az átlagukat, a minimum és a maximum aktivációt. Ezeket az eredményeket a 4.10. ábra szemlélteti. Felülről lefelé helyezkednek el a számok 0-tól 9-ig, és a 0 átlagos aktivációs valószínűségei szerint van rendezve, hogy lássam esetleg szabad szemmel a triviális korrelációkat. Mint látható szabad szemmel semmilyen triviális korreláció nem vehető ki az értékekből, viszont azt érdekesnek találtam hogy a minimum aktivációknál van 1-2 nagyon erős marker ami mindenkorban jelen van az osztályban, ha egy osztályozó ezt a mintázatot megtanulja, akkor már ez alapján is egészen jó eséllyel zárhat ki mintákat ha ezek az aktivációk hiányoznak. Abban is biztosak lehetünk hogy aminek ilyen magas a negatív értéke, azoknak nagyon kicsi lesz a szórása, mert egyszerűen nincsen hova szorniuk, mivel minden neuron legalább egyszer maximális értéket vesz fel. Persze egy valamire való algoritmus magasabb rendű összefüggéseket tanulna meg. Utánajártam a téma művelői között, de nem authorativ szakirodalomban, és azt tudtam meg hogy egy jól működő RBM-nél ilyen véletlenszerűnek látszó aktivációkat kell látni az aktivációk középtérkénel, de valóban nagyon jól disztingváló aktivációk is keletkeznek.



**4.10. ábra.** Az  $RBM$  neuronjainak az aktivációs valószínűségeinek az (a) minimumja, (b) maximumja és (c) középértéke (d) szórása 10 epoch után. Felülről lefelé helyezkednek el a számok 0-tól 9-ig, és a 0 átlagos aktivációs valószínűségei szerint van rendezve.

Hogy ezt egy kicsit tovább vizsgáljam a nullás számra csináltam egy hegedű grafikont is, amit az 4.11. ábra mutat. A sok adat miatt nem lett egészen hegedű grafikon alakja, de nagyon jól szemlélteti amit látni szerettem volna. Van pár neuron aminek az aktivációja teljesen véletlenszerű az egész tartományban elterül, de van olyan is ami nagyon jól lokalizált. Sőt, nem egy neuron van ami egy pontba ömlik össze, tehát a nullás szám minden esetében 1-et vagy 0-át vesz fel.



**4.11. ábra.** Az RBM aktivációinak a hegedű grafikonja a nullás számra vetítve.

**Naiv CIFAR** Miután az MNIST eredményeken látszik hogy az RBM által eszközölt dimenziócsökkentéssel összekötött jellemző kiemelés lerövidítette a számítási időt, és a nyers pixeleken alkalmazott SVM-hez képest nem rontott az osztályozás pontosságában elvégeztem ezeket a méréseket a CIFAR adathalmazon is. Ott kiábránító eredményeket kaptam, az alábbi táblázat közli őket. 4.13.. Mint látszik ez a 90%-os teszt hiba elfogadhatatlan. Már tanítás közben gyanus volt hogy a 139.0-as pontos rekonstrukció hiba túl nagy az mnisten tapasztalt 0.17-0.12-höz képest. Hosszas lamentálás után rájöttem hogy a probléma ott van hogy amíg az MNIST adahalmaz normalizálva van, a CIFAR-ra ez nem elmondható ott 0 és 255 között terjednek az értékek.

**4.13. táblázat.** Az  $RBM + SVM$  kombináció eredménye az CIFAR adathalmazra különböző kernelekkel.

A kernel	RBM mérete	Teszt szet hiba százalék
poly coef0=1 degree=3	763	90%
poly coef0=1 degree=3	261 + 261 + 261	90%

**Normalizált CIFAR10** Miután normalizáltam az adatokat a rekonstrukciós hiba leesett a várt értékre. És a klasszifikációs eredmények is nagyon sokat javultak, ezt a 4.14. táblázat mutatja. Érdekes megfigyelni hogy a színek szerint szétszedett SVM lényegesen jobban teljesített mint az egyben tanított, pedig ugyanannyi neuront használtak és ugyanaddig tanítottam őket. Mindkét struktúra esetében jelentős teljesítmény javulást értem el a nyers adatokon tanított SVM-hez képset, egyenként kb 2 óra alatt futottak le a 7 helyett. Sőt ha az olvasó visszaemlékszik a nyers SVM teljesítménye 55% hibaszázalék volt, ettől az egyben tanított SVM alig marad le 1%-al, a szétdarabolt pedig 4%-al jobban teljesít.

**4.14. táblázat.** Az  $RBM + SVM$  kombináció eredménye az CIFAR adathalmazra különböző kernelekkel.

A kernel	RBM mérete	Teszt szet hiba százalék
poly coef0=1 degree=3	763	56%
poly coef0=1 degree=3	261 + 261 + 261	51%

**A hibrid architektúra értékelése** Úgy gondolom hogy a jelentős sebességnövekedés, és ha jól ki van alakítva akkor a klasszifikációs teljesítmény növekedése indokoltá teszi az ilyen struktúrák alkalmazásának megfontolását egyes szcenáriókban.

#### 4.9. A CNN és az SVM+RBM összehasonlítása

A neurális hálózatok és a kernel metódusok között van egy ciklusosság hogy éppen melyiknek a kutatása jár előrébb, és melyiket érdemes használni. Én személyes érdeklődésből hoztam be a dolgozatomba az RBM + SVM kombinációját, ami sokáig egy nagyon jó klasszifikátornak számított, de talán mondhatjuk hogy eljárt felette az idő. Egy fél meddig épkézláb konvoluciós hálózat a CIFAR-10 osztályozásának a problémáját kevesebb erőforrásból és kevesebb idő alatt 25%-al jobb eredményt tudott hozni mint az előbb említett hibrid megoldás, és még a legjobb SVM-es eredménynél [19] is fél százalékkal jobb eredményt volt képes hozni. Az [http://rodrigob.github.io/are\\_we\\_there\\_yet/build/classification\\_datasets\\_results.html#43494641522d3130](http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html#43494641522d3130)-en összeszedett listából az látszik hogy az eddig elért legjobb eredmény az adathalmazon 3.47% ami 21.43%-al jobb mint a legjobb SVM-es megoldás. Nagyon meg kellene fontolnom manapság hogy milyen feladatoknál használnék SVM-eket ANN-ek helyett.

#### 4.10. A három megközelítés végső összehasonlítása

Úgy gondolom hogy ha az ember ki tud a tanulás jellege miatt térbeli struktúrákat használni, akkor nagyon indokolt eredményekkel kell alátámasztani hogy az ember a konvoluciós neurális hálózatokkal szemben, egy másik architektúra javára döntsön. Ehhez viszont számolni kell a megnövekedett erőforrás igényekkel tanítás közben. Az MLP-ket úgy tudom elképzelni hogy a közeljövőben egy nagyobb architektúra részeként lehet majd használni, mint például az adaptív aktivációs függvényeknél ahogyan azt a "Learning Activations Functions to Improve Deep Neural Networks" [?] is taglalja. A kernel gépeknek most egy kicsit hidegebb korszakát éljük, mert ezeket a struktúrákat tudtommal lényegesen nehezebb nagyon adathalmazokon tanítani, de úgy gondolom hogy mivel még ezeken a módszereken is nagyon sokan dolgoznak esetleg elképzelhető hogy pár év múlva ugyanakkora népszerűségnek fognak örvendeni mint 2012-ben. De én most nem láttnám indokoltnak hogy ezeket a struktúrákat használjam képosztályozási feladatokra, még fejlesztett lokalitású[19] kernelekkel sem.

## 5. fejezet

# Az eredmények összefoglaló értékelése

**Megszerzett tudás** Ebben a szakdolgozatban úgy vélem hogy az egyetemen megszerzett tudásbázisomra alapozva képes voltam lényegesen tovább bővíteni az ismereteimet a gépi tanulás, ezen felül is a képosztályozás téma körében. A következő ismeretek megszerzését tartom kifejezetten előnyösnek:

- A domináns irányzatok és architektúrák megismerése a neurális hálózatokkal való képosztályozásban.
- Az egyes adathalmazok megismerése amin az ember érdemben össze tudja hasonlítani a munkáját másokéval.
- Több fejlesztési keretrendszer futólagos, és a Tensorflow -mint a neurális hálózatok fejlesztésére alkalmas eszköz- mélyebb megismerése
- Tapasztalat szerzése abban hogy hogyan kell teljesítmény méréseket végezni a hálózatokon, amik egy erőforrás korlátozott alkalmazásnál igen fontosak lehetnek.
- A neurális hálózatok tanítására használt hardwarek lehetőségeinek és korlátjainak megismerése, illetve futólagos érintése az olyan új megoldásoknak mint az nvidia jetson platform.

**A végső konklúziók levonása** Mint azt a dolgozatom elején is említettem, ezeket az eljárásokat azzal a céllal értékeltettem hogy ki, hogy egy valós projektben legyenek majd majdan alkalmazva, ahol videofelvételeken szeretnénk majd objektumokat felismerni egy erőforrás korlátos rendszerben. Arra a konklúzióra jutottam hogy a Tensorflow a megfelelő keret rendszer lenne a céljainknak, mégpedig azért mert igen könnyen tudnánk vele a hálózatokat fejleszteni, finomhangolni, nagyteljesítményű GPU griden tanítani, majd egy erőforrás korlátos beágyazott rendszerre telepíteni. Architektúralis szempontból én úgy gondolom hogy a projektnek legmegfelelőbb alap struktúra a konvoluciós hálózatok lennének, mivel igazán nagy memória és számítási igényük csak tanítás közben van, de utána egy

korszerű GPU-n kb 200 képet tudnak kiértékelni másodpercenként, ami számunkra kielégítő sebesség.

**További munka, és fejlesztési lehetőségek** Az elsődleges célunk az lesz hogy a jelenlegi tudásunkra építve egy prototípus képosztályzó rendszert építsünk a robotrepülőgéphez. Ehhez hozzá tartozik majd további irodalomkutatás a konvoluciós hálókkal való videofeldolgozás területén, ami egy fantasztikusan izgalmas terület. Egy másik potenciális, lényegesen alaputatás jellegűbb irány amivel külső konzulensem foglalkozik. Mint a láthattuk a modern hálózatoknak hatalmas paraméter terük van, és az ebben való optimalizálás komoly problémákat tud okozni algoritmikai komplexitása, és erőforrás igénye miatt. Egy megfelelően választott hibafüggvényel viszont esetlegesen el lehetne érni hogy csak egy kissébb, de azonos tulajdonságokkal rendelkező hálót keljen tanítani.

# Köszönetnyilvánítás

Ez nem kötelező, akár törölhető is. Ha a szerző szükségét érzi, itt lehet köszönetet nyilvánítani azoknak, akik hozzájárultak munkájukkal ahhoz, hogy a hallgató a szakdolgozatban vagy diplomamunkában leírt feladatokat sikeresen elvégezze. A konzulensnek való köszönetnyilvánítás sem kötelező, a konzulensnek hivatalosan is dolga, hogy a hallgatót konzultálja.

# Irodalomjegyzék

- [1] The cifar datasets. <https://www.cs.toronto.edu/~kriz/cifar.html>.
- [2] The mnist dataset. <http://yann.lecun.com/exdb/mnist>.
- [3] Scaling convolutional networks. [https://www.tensorflow.org/versions/r0.11/tutorials/deep\\_cnn/index.html](https://www.tensorflow.org/versions/r0.11/tutorials/deep_cnn/index.html).
- [4] Li Fei-Fei Andrej Karpathy. Deep visual-semantic alignments for generating image descriptions. April 2015.
- [5] T. N. Wiesel D. H. Hubel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. 1962.
- [6] Alex Krizhevsky et al. Imagenet classification with deep convolutional neural networks.
- [7] Alexander G. et al. Learning a deep hybrid model for semi-supervised text classification. 2015.
- [8] Hugo Larochelle et al. Exploring strategies for training deep neural networks. 2009.
- [9] Ian J Goodfellow et al. Generative adversarial nets. June 2014.
- [10] Kaiming He et al. Deep residual learning for image recognition.
- [11] Max Jaderberg et al. Spatial transformer networks. February 2016.
- [12] Ross Girshick et al. Rich feature hierarchies for accurate object detection and semantic segmentation. October 2014.
- [13] Szegedy et al. Going deeper with convolutions.
- [14] Yann LeCun et al. Gradient-based learning applied to document recognition. November 1998.
- [15] Cybenko G. Approximations by superpositions of sigmoidal functions. 1989.
- [16] Geoffrey Hinton. A practical guide to training restricted boltzmann machines. aug 2010. <http://www.cs.toronto.edu/~hinton/absps/guideTR.pdf>.
- [17] Kurt Hornik. Approximation capabilities of multilayer feedforward network. 1991.

- [18] Yoshua Bengio Hugo Larochelle. Classification using discriminative restricted boltzmann machines. 2008.
- [19] Tong Zhang Kai Yu. Improved local coordinate coding using local tangents. 2010.
- [20] Francis Bach Mark Schmidt, Nicolas Le Roux. Minimizing finite sums with the stochastic average gradient. September 2013.
- [21] Rob Fergus Matthew D. Zdziarski. Visualizing and understanding convolutional neural networks. November 2013.
- [22] Google research. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. 2016.
- [23] Kishore Konda Zhouhan Lin, Roland Memisevic. Rumelhart, david e.; hinton, geoffrey e.; williams, ronald j. October 1986.
- [24] Kishore Konda Zhouhan Lin, Roland Memisevic. How far can we go without convolution: Improving fully-connected networks. November 2015.
- [25] Kishore Konda Zhouhan Lin, Roland Memisevic. Wavenet: A generative model for raw audio. September 2016.