Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Távközlési és Médiainformatikai Tanszék

Osváth Márton

KÁRTÉKONY NÖVÉNYEK AZONOSÍTÁSA KÖZÖSSÉGI ADATBÁZISBÓL MÉLY KONVOLÚCIÓS HÁLÓZATOKKAL

KONZULENS

Dr. Tóth Bálint Pál

BUDAPEST, 2016

Tartalomjegyzék

Kivonat

TDK munkám során kártékony növények automatikus felismerésére és azonosítására alkalmas mély konvolúciós neurális hálózatok létrehozásával foglalkoztam. E kártékony fajok irtása mind gazdasági mind egészségügyi szempontból fontos feladat, hiszen természetes ellenség hiányában gyorsan elterjednek veszélyeztetve az adott terület őshonos élővilágát, illetve például allergén fajok esetén az emberi egészséget is.

Az elmúlt évtizedben a gépi képfelismerésnek és képosztályozásnak egyik meghatározó technikájává váltak a mély konvolúciós neuronhálók. A grafikus processzorok (Graphical Processing Unit, GPU) ugrásszerű fejlődése, a sok digitális adat és az új tudományos eredmények lehetővé tették, hogy napjainkra a konvolúciós neurális hálózatok akár még az embernél is jobb eredményeket érjenek el a képek felismerésében.

Kutatási témám ötletét a LifeCLEF 2016-os verseny, azon belül a PlantCLEF 2016-os versenykiírás adta, melynek során közösségi adatgyűjtéssel készített adatbázisból származó képeket kellett a tanító adatokból megtanult osztályokba sorolni, így azonosítva a rajtuk szereplő kártékony növényeket.

A kutatásom a mély neurális hálózatokon alapul. Dolgozatomban megvizsgálom a különböző képfelismerő architektúrák elméleti hátterét és kiválasztom az általam a célnak legmegfelelőbbnek ítélt módszereket. A kutatás során 3 különböző típusú, a nemzetközi szakirodalom szerint nagy pontosságú és széles körben használt konvolúciós neurális hálózatot használok. A neuronhálókat vagy az alapoktól kezdve tanítom, vagy pedig más képekkel előretanított modelleket használok, és ezeket tanítom tovább új klasszifikációs rétegekkel. A tanítást két eltérő teljesítményű GPU-ra optimalizálom.

Az eredményeket mindegyik konvolúciós hálózat esetén két különböző metrikával értékelem ki: pontosság (accuracy) és MAP (Mean Average Precision). Eddigi munkám eredményeként mintegy 70%-os pontossággal meg tudom állapítani, hogy kártékony növény van-e a tesztadatbázis képein.

A PlantCLEF 2016 versenyre készített rendszer angol nyelvű konferencia cikkben került ismertetésre, én ebből kiindulva tovább folytattam a kutatást, eredményeimet dolgozatomban mutatom be.

Abstract

My research is focused on creating convolutional neural networks capable of automated detection and identification of invasive plants. Extermination of these invasive species is crucial for both economical and health reasons, as - by having no competitor species - they can spread rapidly endangering local flora, and in some cases their pollens may trigger serious allergic reactions.

In the last decade convolutional neural networks became one of the most significant automated image recognition and image classification technique. With the dramatical improvement of graphical processors (GPUs), the plenty of digital data and the new scientific results today's convolutional neural networks can surpass man in image recognition.

The idea of this research comes from the LifeCLEF 2016 challenge, PlantCLEF 2016 assignment. The task was to classify test images into classes, identifying the invasive species present using training images from a crowdsourced dataset.

My research is based on deep neural networks. In my study I review different image recognition architectures' theoretical background to find the adequate method for this task. I use 3 types of mainstream and highly accurate (according to literature) convolutional neural networks. I either train the networks from scratch or use pretrained networks and train them further with new classification layers. I optimize the training for two GPUs of different performance.

I evaluate the networks' outcome using two metrics: accuracy and MAP (Mean Average Precision). According to my current results 70% accuracy can be achieved in identifying invasive plants on the test dataset using these networks.

The system made for the PlantCLEF 2016 assignment was described in an English conference paper. In this study I concentrate on my own results continuing the research after the publication of the conference paper.

1. Bevezetés

Jelen dolgozat egy napjainkban egyre fontosabbá váló témát, a kártékony növények automatikus felismerését taglalja. A kártékony növényeken belül is elsősorban az inváziós növények azonosításával foglalkozom. Inváziósnak nevezzük azokat a növényeket, melyek nem őshonosok az adott környezetben, így nincs természetes ellenségük, hirtelen képesek elterjedni, felborítva az addigi ökológiai egyensúlyt. Hazánkban talán a leginkább ismert ilyen növény az ürömlevelű parlagfű (*Ambrosia artemisiifolia*), de összesen mintegy hetven kártékony növényfaj van jelen Magyarországon. Mielőbbi azonosításuk és irtásuk rendkívül fontos feladat, hiszen kipusztíthatják az őshonos növényeket, gazdasági károkat okozhatnak, továbbá pollenjeik tömegesen válthatnak ki akár különösen súlyos allergiás reakciókat.

Ma már több olyan okostelefon alkalmazás is van (pl. iNaturalist[1], Tela Botanica[2]), melyek segítségével bárki tölthet fel képeket a környezetében található növényzetről, külön megjelölve, hogy a képen szereplő növény kártékony-e. Ezzel a közösségi („crowdsourced”) adatgyűjtéssel mára már jelentős méretű kép adatbázis áll rendelkezésre inváziós növényekről. Viszont hatalmas kihívás egy ilyen képi közösségi adatbázissal dolgozni, mert nem előre meghatározott szempontok alapján, profi fotósok állítják be a képeket, hanem hétköznapi emberek eltérő irányokból és szögekből, eltérő készülékekkel különböző minőségű képeket készítenek. Sőt, akár egy képen több növényrészlet is előfordulhat, ezzel tovább nehezítve a felismerést.

Kutatási témám ötletét a LifeCLEF 2016-os verseny, azon belül a PlantCLEF 2016-os versenykiírás adta. A versenyben megfogalmazott feladat célja az volt, hogy 113205 darab, 1000 osztályba sorolt, az előbb említett „crowdsourced” adatbázisból származó tanító adatra támaszkodva további 8000 tesztképet kellett minél nagyobb pontossággal a tanító adatokból megtanult osztályokba sorolni. A feladatot mély neurális hálózatokkal oldottam meg.

Bár a neurális hálózatok alapjait már az 1960-as években lefektették, a gépi képfelismerés és képosztályozás egyik meghatározó technikájává csak az elmúlt évtizedben

[1] http://www.inaturalist.org/
[2] http://www.tela-botanica.org/site:accueil?langue=en

váltak. Ennek oka, hogy a korai fázisban még nem állt rendelkezésre megfelelő mennyiségű digitális adat és elég erőforrás ahhoz, hogy az akkori egyéb technikáknál hatékonyabb modellt lehessen készíteni a segítségükkel. A grafikus processzorok (Graphical Processing Unit, GPU) ugrásszerű fejlődése, a sok digitális adat és az új tudományos eredmények mára már lehetővé tették, hogy a konvolúciós neurális hálózatok bizonyos körülmények mellett akár még az embernél is jobb eredményeket érjenek el a képek felismerésében.

A képfelismerés mellett a neurális hálózatok további feladatokat is elláthatnak. Például komplex tünetegyüttesek alapján nagy pontossággal meg tudják állapítani egy ember betegségét (IBM Watson). De a legkülönfélébb klasszifikációs feladatokra is alkalmasak lehetnek. A neurális hálózatokban azért rejlik hatalmas potenciál, mert segítségükkel olyan feladatok is megoldhatóvá váltak, amelyek eddig rengeteg programozást igényeltek volna.

1.1. PlantCLEF 2016

A verseny célja ismeretlen növények klasszifikációja ismert növények képei alapján. 2010 óta a LifeCLEF verseny (régebben ImageCLEF) részeként kerül évente megrendezésre a PlantCLEF kategória. A versenyhez tartozóan minden évben tartanak egy konferenciát: CLEF (Conference and Labs of the Evaluation Forum). Az eddigi években csak a növények fajokba sorolásáról szólt a verseny, idén először viszont külön bevonták a kártékony növények felismerését is. Ez a versenyen abban nyilvánult meg, hogy más metrikával értékelték ki az eredményeket, és külön kiszámolták a felismerés eredményét a kártékony növényekre.

A versenyen csapatban indultunk: Dr. Szűcs Gábor és Papp Dávid a szupport vektor gépeket (Support Vector Machine, SVM) használva közelítették meg a problémát, konzulensem, Dr. Tóth Bálint Pál és én pedig mély neurális hálózatokat alkalmaztunk. A munkánk folyamatáról konferencia cikk is született. [1] Ebben a dolgozatban elsősorban a cikk írása során szerzett tapasztalataimat és a cikk utáni munkámat fogom bemutatni.

A verseny keretében elvégzett munkám a többi csapattagtól jól elkülöníthető volt. Dolgozatomban azon eredményekről számolok be, amelyeket a csapat részeként, de a többiektől függetlenül értem el.

2. Célkitűzés

A kutatómunkám célja, hogy a PlantCLEF 2016 versenyen a rendelkezésre álló tanító növény-képek alapján minél jobb eredménnyel osztályokba soroljam a külön kiadott teszt képeket konvolúciós neurális hálózatok segítségével. Ehhez áttekintem és elsajátítom a konvolúciós neurális hálózatok elméleti alapjait, tanítási módszereit. Tanulmányozom a legnépszerűbb, ILSVRC versenyen [2] legjobb eredményeket elérő megoldásokat és ezek implementációit. Az általam meghatározott feltételek alapján kiválasztom a célnak legjobban megfelelő architektúrákat. A hálózat architektúrákat figyelembe véve eldöntöm, hogy előtanított hálót használjak-e vagy tanítsam alapjaitól kezdve a hálót.

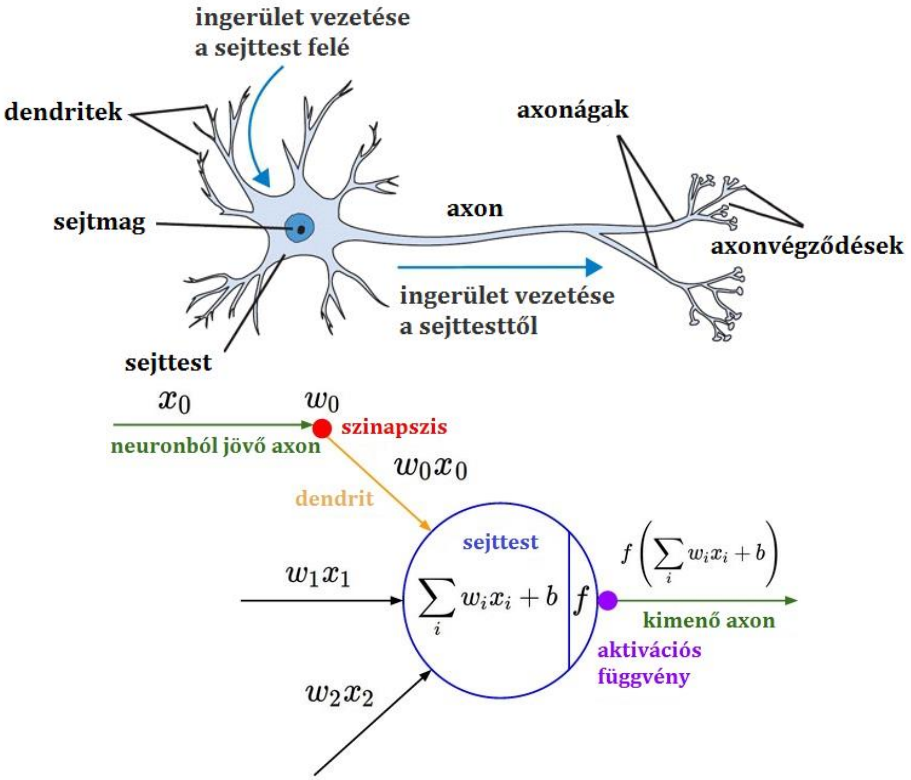
Az adatokat előkészítem („előfeldolgozom”) a kiválasztott hálózatoknak. Optimalizálom a hálózatokat a rendelkezésre álló szoftver és hardver környezetre, különös tekintettel a GPU-n történő számításokhoz. Betanítom a hálózatokat, majd optimalizálom a tanulási hiperparamétereket a minél jobb eredmény elérése érdekében. Végezetül kiértékelem az eredményeket többféle módszerrel (top-1 pontosság, MAP, tévesztési mátrix (angolul: confusion matrix) validációs pontossága). A kiértékelés során a 2015-ben kiadott teszt adatokat használom, melyeknek címkéi is rendelkezésre álltak.

3. Irodalomkutatás

3.1. A neurális hálózatok rövid története

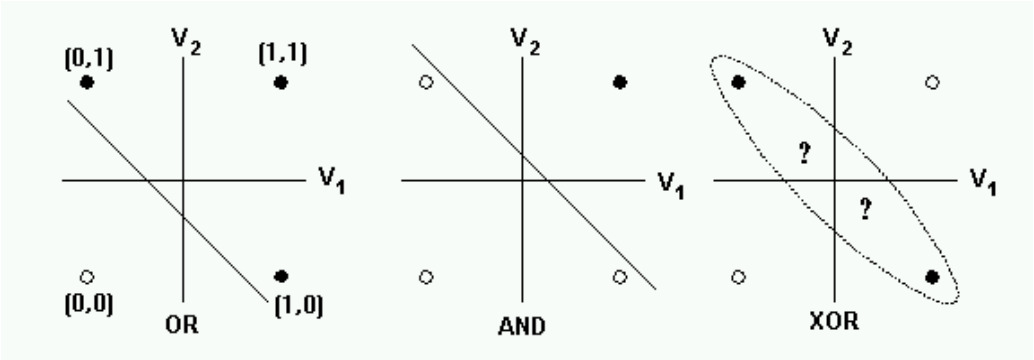
Neurális hálózatok alapvető építőelemét a perceptront (a későbbiekben mesterséges neuron, vagy csak egyszerűen neuron) már az 1960-as években felfedezték. A perceptron egy olyan számítási egység, mely csak egy funkciót lát el. Rendelkezik egy súllyal, és egy nemlineáris aktivációs függvénnyel. [3]

A kapcsolódó források szerint a jórészt biológiai kutatások eredményeképpen merült fel az a gondolat, hogy a természetes, "biológiai" neurális hálózatok mintájára is létrehozhatók számító rendszerek (1. ábra). A sikeres kezdeti próbálkozások után hamar problémákba ütköztek. A legegyszerűbb logikai kapukat (ÉS/VAGY/NEGÁLÁS) képesek voltak megtanulni ezek a hálók, azonban a bonyolultabb lineárisan nem szeparálható problémákat (pl.: XOR) nem tudták megoldani (2.ábra). [4][5]



1. ábra: *Az agyban lévő neuron (fent) és a matematikai modell (lent) összehasonlítása (forrás: Stanford University[3] alapján)*

[3] http://cs231n.github.io/neural-networks-1/#bio

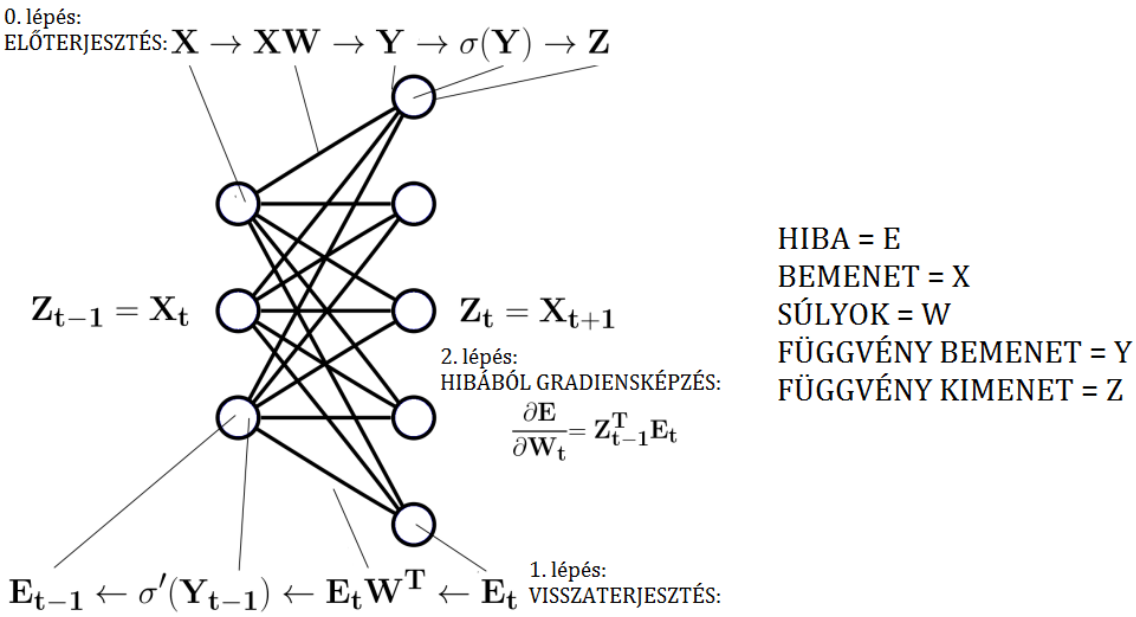


2. ábra: *Lineárisan szeparálható ÉS,VAGY és lineárisan nem szeparálható KIZÁRÓ VAGY (forrás: University of Colorado Boulder[4])*

Eredmények hiányában a kezdeti reményteljes lelkesedés alábbhagyott, és jó időre háttérbe szorultak az ilyen típusú kutatások. Korábban rengeteg támogatást fordítottak a kutatásokra, mégis megakadtak, ezért újfajta szemléletmódra volt szükség. Ez az áttörés egésze 1986-ig váratott magára, amikor Geoffrey Hinton és társai az 1970-es években feltalált hibavisszaterjesztés algoritmust alkalmazták a neurális hálózatok tanítására. [6] Ezzel az algoritmussal taníthatóvá váltak a többrétegű neurális hálózatok. Az algoritmus alapja az először Leibniz által leírt láncszabály. Ennek lényege, hogy kiszámoljuk a hibatagokat az összes rétegre, majd ezekből gradiens vektorokat képzünk, és "frissítjük" a neuronok súlyait a régi súlyuk és a tanulási rátával szorzott gradiensek különbségeként. Az eljárás működésének az alapjait a 3. ábra mutatja be. A tanulási ráta a tanulás sebességére van hatással.

Mivel ez az algoritmus mátrixműveletekkel hatékonyan végezhető, rendkívül jól használhatók a videókártyák (Graphical Processing Unit, GPU), melyeknek teljesítménye az utóbbi 10 évben óriási növekedést mutatott. [7]

[4] *http://ecee.colorado.edu/~ecen4831/lectures/xor2.gif*



3. ábra: *A hibavisszaterjesztés 3 lépése (forrás: Nvidia devblogs[5] alapján)*

3.2. SGD

A legrégebbi és a mai napig egyik leginkább elterjedt optimalizáló, amit a hibavisszaterjesztéssel alkalmaznak, a gradiens módszer, azaz a gradient descent. Ezzel lehet keresni a többdimenziós hibafüggvény minimumhelyét. Képletesen úgy lehet elképzelni, mint egy ködös hegyet, melyen csak 10 méteres körben látunk, és legalacsonyabb helyre szeretnénk eljutni. A gradiens módszer kijelöli a legmeredekebb lefelé vezető utat, majd arra haladunk egy kicsit. A haladás nagyságát a tanulási ráta mellett a batch mérettel is befolyásolhatjuk, mellyel megadjuk, hogy egyszerre hány tanító adatra végezzünk hibavisszaterjesztést (és ezáltal a súlyfrissítést). Háromféle verzió lehetséges, az első az úgynevezett batch learning, amikor az összes tanító adatot egyszerre ráadjuk a bementre, és csak egyszer csinálunk hibavisszaterjesztést. Ilyenkor túlságosan „kisimulhat" a gradiens, így nem találhatjuk meg hatékonyan a minimumpont irányát. A második az ún. „online" learning, ilyenkor egyesével adjuk rá a bemenetre a tanító adatot, majd erre számolunk egyesével hibát és ezt terjesztjük vissza. Ezzel az a probléma, hogy egy mintára („pontra") nem lehet gradiensvektort illeszteni, így nem találhatjuk meg a többdimenziós, szabálytalan hibafüggvény minimumpontja felé vezető irányt. A harmadik pedig a

[5] https://devblogs.nvidia.com/parallelforall/wp-content/uploads/2015/12/backprop-624x659.png

gyakorlatban legelterjedtebb módszer, a “mini-batch” learning, ilyenkor általában véletlenszerűen kiválasztott tanító adatok egy kisebb halmazát adjuk a bemenetre, és a kimenetre számolt hibát visszaterjesztjük, a gradienst pedig átlagoljuk. A másik két módszerrel szemben így gyakorlatban általában nagyobb hatékonysággal találhatjuk meg a minimumhelyeket. Az ilyen, mini-batch-el tanított gradiens módszert sztochasztikus gradiens módszernek nevezzük (a későbbiekben SGD, ami a Stochastic Gradient Descent rövidítése). [8]

Ehhez az optimalizációhoz készültek konvergencia segítő megoldások. Ilyen például a momentum módszer. A gradiens általában váltakozó irányokban csökken, ahogy próbálja megtalálni a hibafüggvény minimumpontját, viszont van egy irány, amelyben általánosan csökken, a momentum segítségével ezt az irányt lehet megtalálni, ezzel segítve a konvergenciát. [9]

Az SGD esetén gyakran használt konvergencia segítő megoldás még a weight decay (felejtő tanulás) és learning-rate decay. A weight decay (amire L2 regularizációként is szoktak hivatkozni) egy mód arra, hogy „kordában tartsuk” a neuronok súlyainak növekedését és csökkenését. Úgy működik, hogy a költségfüggvényt módosítja egy új tag hozzáadásával, melyben figyelembe veszi a súlyok kettes normáját. A weight decay azért fontos, mert így kisebb súlyértékekkel is elérhetjük ugyanazt az eredményt, sőt még a zajosabb adatokat és képes ellensúlyozni, csökkenti a túltanulást, redundánsabb lesz a rendszer. [10] A learning-rate decay pedig arra való, hogy az epoch szám növekedésével arányosan csökkenti a tanulási rátát, ezzel csökkentve a hiba oszcillálását. Egy epoch-nak nevezzük azt, amikor az összes tanuló adat egyszer áthaladt a hálózaton előre, majd egyszer vissza a hibavisszaterjesztés közben. [28]

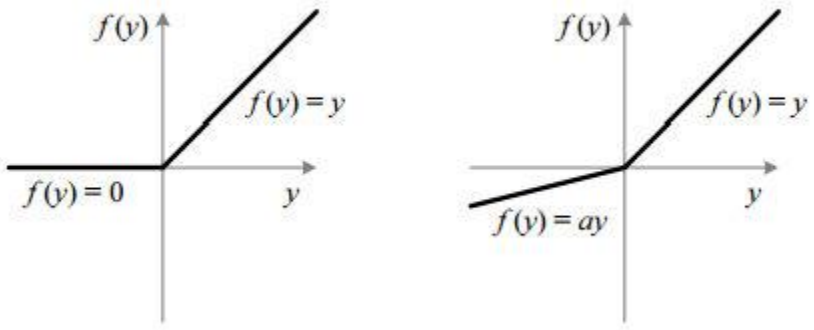
3.3. Adadelta

A másik optimalizáció, amit használtam, az Adadelta optimalizáció. Ez az SGD-n alapul, előnye, hogy itt nincs szükség tanulási ráta beállítására, finomhangolására. Ugyanis ezt és más tanulási paramétereket adaptívan állít be az Adadelta a gradiensek tanulás közbeni alakulása alapján. [11]

3.4. Aktivizációs függvények

A neuronok aktivizációs függvényeként többnyire ReLU-t használtam, ez a Rectified Linear Unit rövidítése. A ReLU egy kis erőforrásigényű függvény, ha negatív bemenetet kap, akkor 0 a kimenete, ha pedig pozitívat, akkor visszaadja a bemeneti értéket (4. ábra). Lehetővé teszi a gyorsabb tanulást más aktivizációs függvényekhez képest, mert vele gyorsabb a gradiens áramlás. Például a korábban használt Sigmoid függvénynél nagyon kicsik a gradiensek a két szélső értékénél, így ott szinte megáll a tanulás. Továbbá, ReLU esetén nincs szükség inicializálásra sem. [12]

A másik aktivizációs függvény, amit használtam a PReLU (Parametric Rectified Linear Unit), mely a ReLU egy módosított változata. Az eltérés az, hogy a negatív tartományban sem konstans a függvény, hanem egy alpha (a) paraméterrel rendelkező lináris. Az 'a' paramétert a PReLU megtanulja a gradiens módszer közben. A PReLU - a vonatkozó cikk szerint - javítja a tanulás hatékonyságát úgy, hogy nem nullázza ki a nem aktív neuronokat, hanem negatív értékeket ad nekik. Továbbá, szinte alig igényel nagyobb számítási kapacitást, mint a ReLU. [13]



4. ábra: *A ReLU (balra) és PReLU (jobbra) függvények közötti különbség (forrás: [13])*

3.5. Dropout

Egy további regularizációs eljárás, amit a munkám során használtam, az ún. dropout. Ez úgy működik, hogy a dropout réteg előtt elhelyezkedő neuronok adott százalékát nem veszi figyelembe a tanulás során. Tehát lényegében fegyelmen kívül hagyja az aktuálisan véletlenszerűen kiválasztott neuronokat és a hozzájuk tartozó kapcsolatokat a többi neuronnal. Minden egyes epoch-nál más és más neuronokat mellőz, így minden epoch-nál új

hálózati architektúra jön létre. Kiértékelésnél viszont nincs dropout, így olyan, mintha több különböző háló aggregált döntését vennénk figyelembe az osztályozásnál. Ez redundánsabbá teszi a hálót. [14]

3.6. A ClassNLL költségfüggvény

A költségfüggvény az, amivel a tanítás kimenetét értékeljük ki. Ilyen a ClassNLL (negative log likelihood n osztályra) költségfüggvény, melyet n osztályú klasszifikicáiós problémákra fejlesztettek ki. Használatához szükség van arra, hogy a hálózat kimenetén logaritmikus valószínűségek jelenjenek meg. Erre alkalmas a LogSoftMax réteg, melyet az utolsó réteg után, a hálózat végére illesztenek. Például, ha 2 osztályba sorolunk képeket, akkor a hálózat kimenetén 2 valószínűség természetes alapú logaritmusa jelenik meg. A kimenet első értéke megmutatja, hogy mekkora valószínűséggel tartozik az első osztályba a bemeneti adat, a második érték pedig a második osztályra adott valószínűség. Fontos, hogy a kimeneti valószínűségek összege mindig 1. [29]

3.7. Batch Normalization

Batch Normalization egy tanítást gyorsító megoldás. Mély (több mint 20 rétegből álló) hálók esetén, ahogy a rétegek súlyai változnak, változnak a rétegek kimeneti értékeinek eloszlásai is. Ahogy egyre mélyebb rétegekhez jutunk, egyre nagyobb ez a változás, ezért csak kis tanulási rátákkal taníthatunk. A batch normalization erre jelent megoldást, úgy, hogy a rétegek közötti adatokat mindig 0 várható értékűre és egységnyi szórásra skálázza. Ezzel jelentősen csökken a tanulási idő, és használhatunk nagyobb tanulási rátákat. [15]

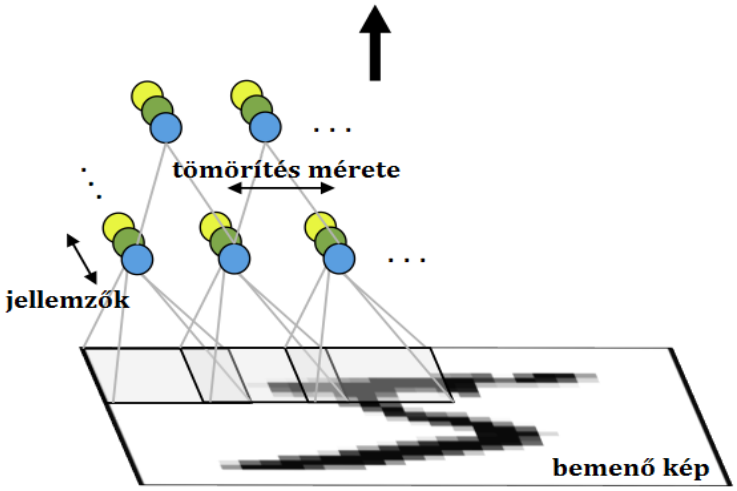
3.8. Konvolúciós neurális hálózatok

Egy konvolúciós neurális hálózatot alapvetően háromféle réteg alkot. Az első típus az úgynevezett konvolúciós réteg, melyről ez a fajta hálózat a nevét is kapta. Ez elsősorban képfelismeréskor használható a tanító adatbázisban lévő képek jellemzőinek tanulására (feature learning). Lényege, hogy a konvolúció kétdimenziós függőségekből (például egy szín csatorna) háromdimenziós neuron architektúrákat készít. Ezekben az architektúrákban az ún. megosztott súlyoknak köszönhetően sokkal kevesebb

összeköttetést használ, mintha minden neuron mindegyikkel össze lenne kötve, mégis hatékonyabban gyűjti ki a képi (térbeli) jellemzőket. A megosztott súlyoknak köszönhetően a gyakorlatban a neuronok akkor aktiválódnak, ha valami olyat "látnak", amit már megtanultak, például egy bizonyos orientációjú élt. Így jönnek létre a konvolúciós "szűrők", például arcfelismerés esetén az alacsonyabb rétegekben csak éleket a továbbiakban arcrészleteket, a felsőbb rétegekben pedig már teljes arcokat tanulnak (6. ábra).

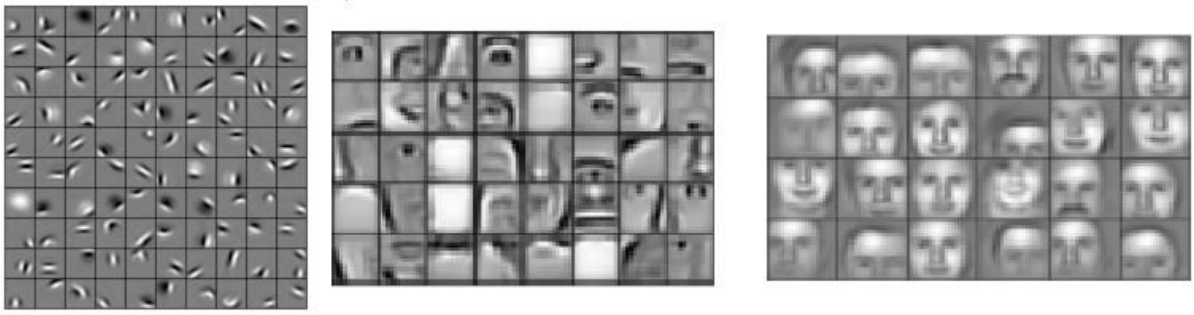
A konvolúciós réteg után opcionálisan következhet egy úgynevezett tömörítő réteg (pooling), mely, mint ahogy a neve is mutatja, több pixelből csinál kevesebbet. A gyakorlatban ez dimenzió szám csökkenésként nyilvánul meg a legfontosabb információk megtartása mellett. Egyik elterjedt típusa az úgynevezett maxpooling, mely esetén például a legsötétebb pixeleket választja ki az algoritmus, ezeket tekinti a legfontosabbnak. Ezt a két rétegtípust annyiszor ismételgetjük, ahányszor az adott aktuális feladat igényli, erre persze csak sejtéseink, heurisztikák lehetnek. Ha túl sokszor ismételjük, akkor túl komplex lesz a hálózat, túl sok hardver erőforrást és adatot fog igényelni és nagyon nehéz lesz tanítani. Ha viszont túl kevésszer, akkor nem lesznek elég nagyok a jellemzők, amik alapján az osztályozás történik.

A végső tömörítő rétegek kimenete össze van kötve az osztályozó egységgel, melynél rétegenként minden neuron mindegyikkel össze van kapcsolva (fully connected layer). [16]



5. ábra: *A neuronok elhelyezkedése egy konvolúciós hálózat esetén (forrás: Standford University[6] alapján)*

[6] http://ufldl.stanford.edu/tutorial/images/Cnn_layer.png



6. ábra: *Egy konvolúciós hálózat "szűrő" rétegei (forrás: Nvidia Devblogs[7])*

3.9. Konvolúciós neurális hálózat architektúrák

Az első jelentős, gyakorlatban is használható konvolúciós neurális hálózat az 1995-ös LeNet volt, mely kézzel írott számok, elsősorban amerikai irányítószámok felismerésére volt képes. Ez a háló az alapja szinte minden ma használt konvolúciós neurális hálózatnak. Érdekes megjegyezni, hogy az akkori technológiával egy ilyen, ma már egyszerűnek tűnő probléma betanítása 2-3 hetet vett igénybe. [17]

A következő történelmi jelentőségű konvolúciós neurális háló, az az Alex Krizhevsky és Geoffry Hinton által kidolgozott AlexNet. Ez a háló az egyik leghíresebb, minden évben megrendezett képfelismerő versenyre az ILSVRC-re (ImageNet Large Scale Visual Recognition Challenge) [2] készült és meg is nyerte azt 2012-ben, szignifikáns mértékben megelőzve a korábbi technikákkal (pl. szupport vektor gép) készült versenytársait. A versenyről az érdemes tudni, hogy a tanító képek az ImageNet [18] online adatbázisból kerültek ki, mely több mint 15 millió címkézett, 22 ezer osztályba sorolt, jó minőségű képet tartalmaz. A versenyen az imént említett adatbázisból 1.2 millió 1000 osztályba sorolt tanító kép alapján kell 5 tippet adni minden tesztképre. 2012-ben neveztek először a versenyre konvolúciós neurális hálózattal és azóta a neurális hálózatok sikere töretlen ezen a versenyen (7. ábra).

Az AlexNet annak köszönheti a győzelmét, hogy sok új ötletet tartalmazott, és a fejlesztői technikailag megoldották, hogy párhuzamosan két GPU-n lehessen tanítani (8. ábra), így dupla annyi erőforrást tudtak használni a modell építéséhez és tanításához. Aktivációs függvénynek mindenhol ReLU-t használtak, 0,5-ös dropout-ot állítottak be az

[7] https://devblogs.nvidia.com/parallelforall/wp-content/uploads/2015/11/hierarchical_features.png

utolsó klasszikációs rétegnél, és SGD-vel tanítottak 128-as batch mérettel. A túltanulást azzal csökkentették, hogy a tanítóadatokat "dúsították" úgy, hogy 224x224 pixeles részeket vágtak ki véletlenszerűen a 256x256 pixeles képekből majd tükrözték is őket vízszintesen. Így 2048-szorosára növelték a tanító adatok számát, ezzel elegendő adat állt rendelkezésre a 7 rejtett réteg tanítására, ami akkoriban igen soknak számított. Az AlexNet 37,5%-os top-1 és 17%-os top-5 hibaarányt ért el a versenyen. [19]

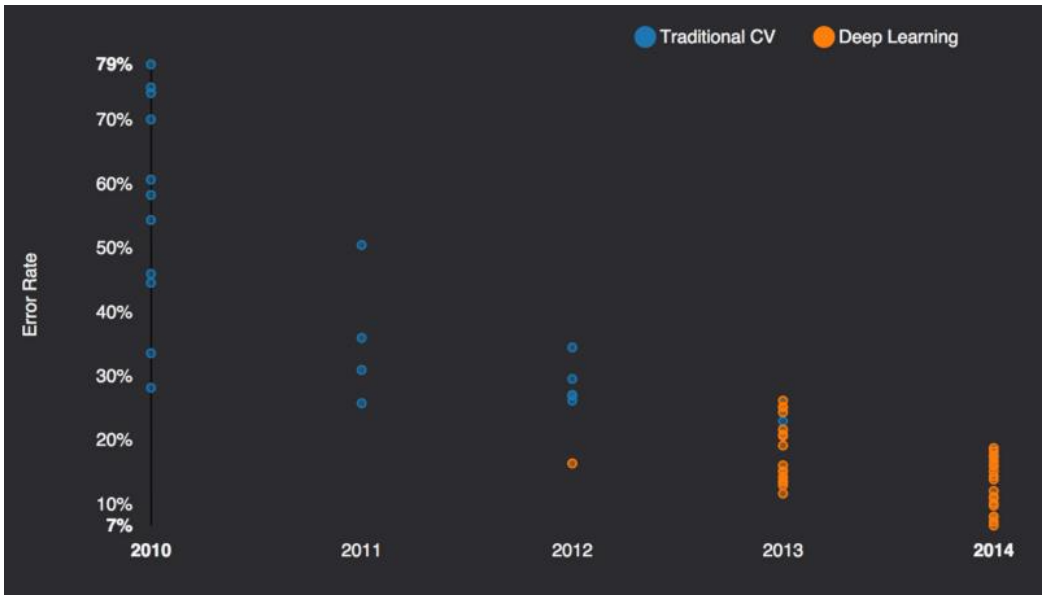
A top-5 hibaarányt a következő képlet alapján számolhatjuk:

$$e = \frac{1}{n} * \sum_{k} min_i d(c_i, C_k), \tag{1}$$

ahol n az osztályok száma, ci a jósolt címke, Ck a valós címke.

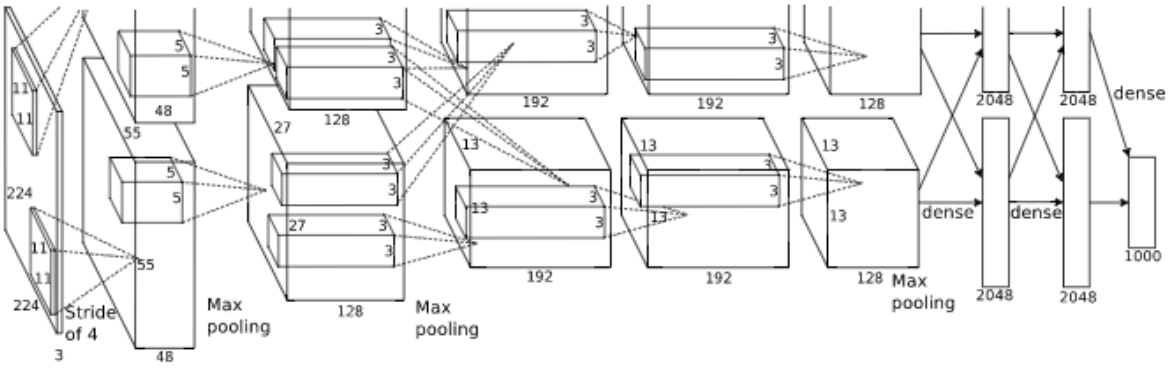
A d függvény pedig a következőképpen adódik:

$$d(a, b) = \begin{cases} 0 \; ha \; a = b \\ 1 \; egyébként \end{cases} \tag{2}$$



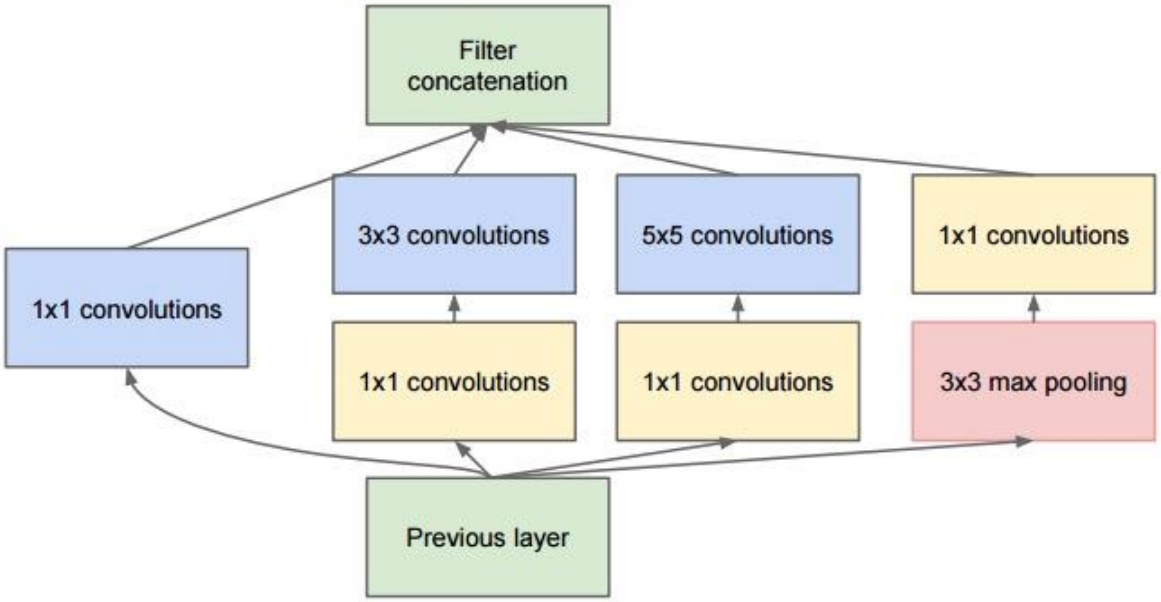
7. ábra: *Az ILSVRC versenyen a klasszifikációs hiba javulása 2010-2014 és a Deep architektúrák megjelenése és elterjedése (forrás: Musings on Deep Learning[8])*

[8] https://cdn-images-1.medium.com/max/800/1*k0b39xf47de-Bqr9KcK9hw.png



8. ábra: *Az AlexNet felépítése 2 GPU-s módban (forrás: [19])*

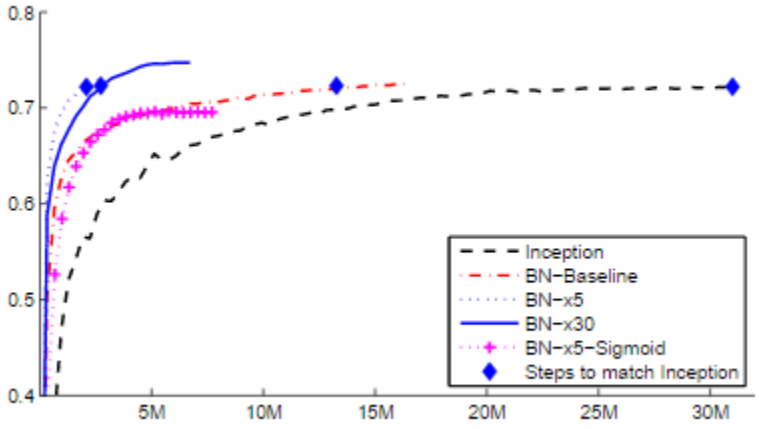
A munkám szempontjából fontos további hálók, a Google által kifejlesztett Inception hálók. Az első verzió 2014-ben nyerte meg LSVRC versenyt. A tanító képekkel az AlexNethez hasonlóan jártak el, csak itt négyféle skálázást készítettek a képekből és ezekből vágtak ki szisztematikusan 224x224 pixeles négyzeteket és ezeket is tükrözték, ezzel növelve a tanító adatok számát. E hálók nagy újítása az Inception modul, mely első ránézésre 1x1, 3x3 és 5x5-ös konvolúciós filterek párhuzamos összekapcsolásának tűnhet (9. ábra), azonban ennél bonyolultabb az elméleti háttere. [20]



9. ábra: *Egy Inception modul felépítése (forrás:[20])*

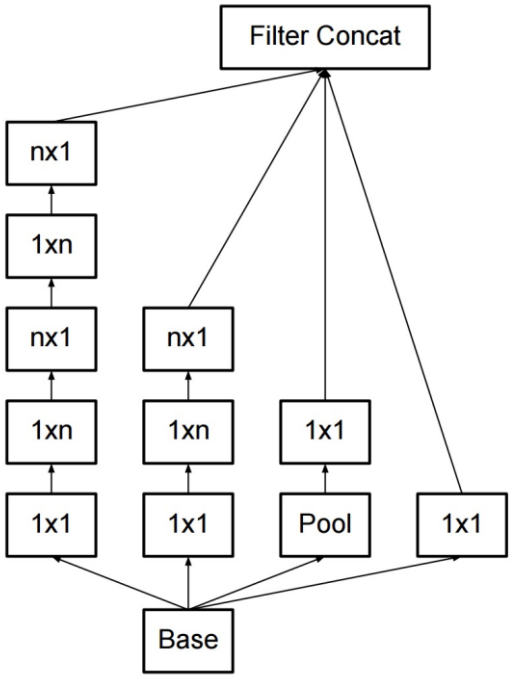
Az Inception modul talán legnagyobb újítását az 1x1-es konvolúciós rétegek jelentik, melyek lecsökkentik a képekből kiszedett jellemzőket a sokkal számításigényesebb 3x3-as és 5x5-ös konvolúciós rétegek előtt. Ezeket gyakran NiN-nek (Network-in-network) is hívják, mert egy kis neurális hálózatként képzelhetők el. A NiN-ek összetett csoportokat hoznak létre a képi jellemzőkből - ezzel jelentősen (mintegy tizedére) csökkentve a hálózat erőforrásigényét. Egy gyakorlati példában bemutatva, ha 256 jellemző megy be és jön ki egy Inception blokkból, és ezt csak 3x3-as konvolúciós réteggel akarnánk megoldani, az 256x256x3x3, azaz kb. 600 ezer művelet lenne. Ezért inkább először a jellemzők negyedét konvolváljuk 1x1 konvolúcióval (256x64x1x1 művelet), majd a 64 jellemzőt 3x3-as konvolúcióval, 64x64x3x3, és 1x1-es réteggel visszaállítjuk a 256 jellemzőt: 64x256x1x1. A műveletek számát összeadva így mintegy 70000 műveletet kapunk. Ráadásul bizonyítottan nem is vesztünk el jellemzőket a képből. Ez a módszer tette lehetővé, hogy ez a hálózat 22 rétegből álljon. Ekkora méretű hálót a korábbi módszerekkel már jellemzően nem lehetne tanítani. Az Inception hálózattal 6,67%-os top-5-ös hibaarányt értek el a 2014-es LSVRC versenyen [20].

A következő, munkám szempontjából fontosnak tekintett háló az Inception V2, melyről nem is készült külön cikk a fejlesztőktől, mert ahhoz - meglátásuk szerint - nem tartalmazott elég újdonságot az első verzióhoz képest. Az Inception háló fejlesztői találták ki viszont a korábbiakban bemutatott Batch Normalization-t, és ezt tesztelték az Inception háló első verzióján, 2015 februárjában ejtettek még néhány architektúrális módosítást, például az 5x5-ös konvolúciós réteget kicserélték két 3x3-as rétegre minden Inception blokkban. Ezekkel a módosításokkal 5,82%-ra tudták csökkenteni a top-5 hibaarányt, ráadásul sokkal gyorsabban is tanult be a hálózat, mivel a tanulási rátát egészen 0,045-ig lehetett növelni 0,0015-ről, melyet a Batch Normalization tett lehetővé (10. ábra) [14]



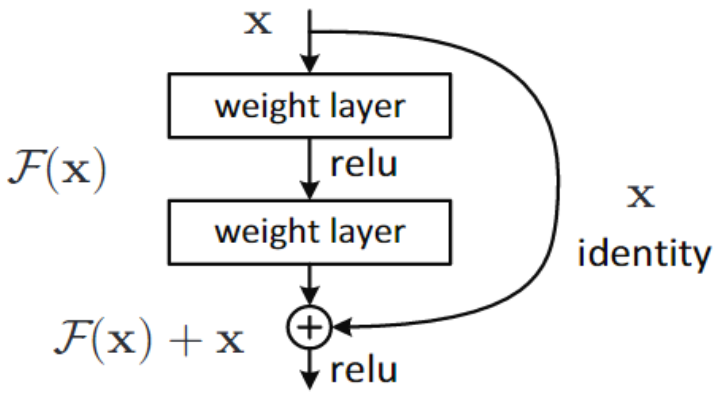
10. ábra: *Az Inception hálók pontossága a tanító lépések függvényében, a BN a Batch Normalization-t jelenti, míg az x után lévő számok jelzik, hogy hányszoros tanulási rátával tanították a hálót, kék rombuszok pedig azt, hogy mikor érték el az Inception háló pontosságát a Batch Normalization-t használó hálók (forrás:[14])*

Ezután következett az Inception háló harmadik változata, 2015 decemberében. Ez a verzió nagy változásokat tartalmazott az Inception modulban. Az Inception V2-ből jött, hogy 5x5-ös konvolúciós rétegeket kicserélték 3x3-asra. Ez a háló már 299x299 pixeles képeket fogadott a bemenetén és 42 réteget tartalmazott. Továbbá, a hálózat közepén bevezették az úgynevezett faktorizált konvolúciót (11. ábra). Ennek lényege, hogy szétszedik a 3D-s konvolúciós réteget csatornánkénti térbeli konvolúcióra és lineáris csatorna vetítésre, több azonos funkciót betöltő egységet egymás után kötnek, ezzel megnövelve az erőforrások kihasználását. Ezekkel a fejlesztésekkel 4,2%-os top-5 hibaarányt sikerült elérni ugyanazon a teszthalmazon, amit eddig is használtak. Érdemes megjegyezni, hogy az emberi osztályozás ezen az adatbázison 5,1% top-5 hibaarány, azaz ezzel a hálózattal meghaladták az emberi felismerést [21][22].



11. ábra: *Az Inception V3 hálózatban használt faktorizált konvolúció sematikus ábrája (forrás:[21])*

A legújabb Google fejlesztés, amelyről 2016. október 10-én jelent meg cikk, az Xception (Extreme Inception). Ahogy a neve is sugallja, ez a háló szintén az Inception hálókból merít ötletet és továbbfejleszti azt. Hasonlóan az Inception V3 hálóhoz, itt is faktorizált konvolúciót alkalmaztak, viszont itt sokkal átláthatóbban és hatékonyabban implementálták ezt a műveletet. Továbbá, reziduális (maradékjel) kapcsolatokat is tartalmaz, melyet a Microsoft fejlesztett ki 2015-ben. A kifejlesztett hálózat a ResNet-152 mely, ahogy a nevéből is sugallja, 152 réteget tartalmazott. A reziduális kapcsolatok lényege, hogy fentebbi réteg kimenete elérhetővé válik több szinttel lentebb is, azaz képes szinteket átugrani az információ. Ennek segítségével gyorsabban és eredményesebben képes tanulni a háló, hiszen, ha nincs több értékelhető „jellemző” a képen, akkor átugorhatja az alsóbb szinteket (12. ábra). Az Xception 36 konvolúciós réteget tartalmaz és körülbelül ugyanannyi paramétert, mint az Inception V3, tehát körülbelül ugyanannyi a számításigénye, mint az Inception V3-nak. Mégis jobb eredményeket ér el (13. ábra), főleg az ILSVRC verseny adatbázisánál jelentősen nagyobb adathalmazokon. [23][24]



12. ábra: *A reziduális hálózat felépítése (forrás:[22])*

	Top-1 accuracy	Top-5 accuracy
VGG-16	0.715	0.901
ResNet-152	0.770	0.933
Inception V3	0.782	0.941
Xception	**0.790**	**0.945**

13. ábra: *Az Xception háló top-1 és top-5 pontossága a többi hálóhoz képest az ILSVRC adathalmazon (forrás: [24])*

E megvizsgált hálók alapján választom ki a feladat szempontjából elméleti megfontolások alapján a legjobbnak ítéltet. A klasszifikációs feladat feltételei alapján kiválasztom az AlexNet-et, mely jó viszonyítási alapnak, mert ez az egyik legkiforrottabb, sok tématerületen kipróbált konvolúciós hálózat. Továbbá az Inception hálókhoz állnak rendelkezésre megfelelő források, implementációk és e források alapján várhatóan jobb eredményeket fognak produkálni az AlexNet-hez képest.

Ezek a hálók mind az ILSVRC versenyre készültek, azaz 1000 klasszifikációs osztályra lettek tervezve, így rendkívül jól idomultak a PlantCLEF versenyhez, ahol szintén 1000 osztályba kell sorolni a képeket.

Az AlexNet architektúrán kicsit csiszolok a tanítások előtt, ezt be fogom mutatni a későbbiekben, míg az Ineception hálózatok szerkezeti felépítésén nem változtatok. Az Inception hálókon a Google mérnökei dolgoztak, a szinte korlátlan hardver erőforrások segítségével finomhangolták ezeket a hálózatokat, melyek tartalmazzák szinte az összes

újítást és ötletet, mely az elmúlt években született. Ezeket az újdonságokat be is mutattam az irodalomkutatásomban.

A kutatás első fázisában az előbb említett architektúrák alapos áttanulmányozása után döntöttem úgy, hogy nem érdemes teljesen új hálózatokat kifejleszteni, erre a feladatra alkalmasak a már létező hálózatok.

4. Módszerek

4.1. Adatbázis

A tanító adatbázis 113205 képből és a hozzájuk tartozó leíró (xml) fájlból állt. A képek közösségi (crowdsourced) adatgyűjtésből származnak. Ez azt jelenti, hogy mindenki számára elérhető mobil alkalmazások segítségével gyűjtött „amatőr" képek, melyeket vállalkozó szellemű egyének készítettek átlagos felszereléssel. Tehát nem profik által megkomponált képekről van szó, hanem változatos felbontásban, nagyításban változó fényviszonyok között és különböző szögekből készült fotókról.

A hozzájuk tartozó leíró fájl tartalma a megfelelő sorrendben:

- ObservationID (megfigyelési azonosító): Az azonos időben, helyen és ugyanarról a növényről készült képeknek ugyan az a megfigyelési azonosítójuk (szám)
- MediaID: Az adott képhez tartalmazó sorszám, ez a kép és a képhez tartozó leíró fájlneve (szám)
- Vote: A kép emberek által besorolt felismerhetősége 1-től 5-ig skálán (szám)
- Content: A kép tartalma (levél, virág, ág, termés, stb.) (14. ábra) (karaktersorozat)
- ClassID: A képen található növény osztálya (1000 féle lehet) (szám)
- Family: A növény családja (karaktersorozat)
- Genus: A növény nemzetsége (karaktersorozat)
- Species: A növény fajneve (karaktersorozat)
- Author: A kép készítőjének neve (karaktersorozat)
- Date: A kép készültének ideje (dátum)
- Location: A kép készültének helye (karaktersorozat)
- Latitude: A kép készültének szélességi foka (szám)
- Longitude: A kép készültének hosszúsági foka (szám)
- YearInCLEF: A kép mióta szerepel az adatbázisban (évszám)
- ObservationID2014: A kép 2014-es megfigyelési azonosítója (szám)
- ImageID2014: A kép 2014-es fájlneve (szám)
- LearnTag: Lehet tanító vagy teszt adat (karaktersorozat)

Ahogy láthatjuk, rengeteg meta adat áll rendelkezésünkre a képekkel kapcsolatban, ezzel segítve a felismerést. Viszont sajnos ezek a meta adatok nem konzisztensek, azaz nincsenek minden képnél kitöltve és számos esetben pontatlanok.

Rendelkezésre álltak még a 2015-ös versenyen kiadott azonosítandó képek a hozzá tartozó leíró fájlokkal, mely tartalmazta a ClassID-t is. Ezek ideálisnak bizonyultak a saját tesztjeimhez. A 2016-ra kiadott azonosítandó képek esetében pedig a ClassID üresen volt hagyva, hisz a klasszifikáció volt a verseny célja. Minden 2016-ban kiadott képre 1000 darab valószínűséget kellett adni az 1000 db osztályra.

14. ábra: *Példa a képek tartalmára (Forrás: [25])*

4.2. Rendelkezésre álló szoftver és hardver környezet

Minden szkriptet és neurális hálózatot Lua nyelven valósítottam meg, mely egy igen régi programnyelv és a Python-hoz hasonlóan futásidőben fordítódik. Azért ezt a nyelvet

választottam, mert könnyű jól áttekinthető, letisztult kódot készíteni benne, és ezt a nyelvet használja a Torch7 mély tanuló keretrendszer. A Torch7 [26] egy olyan Lua nyelven megírt könyvtár, melynek eszközkészletében megtalálhatók a legkülönfélébb neurális hálózatok rétegei, melyekből szabadon építhetők hálózatok. Torch7-el igen jól paraméterezhető, alapjaitól felépített hálókat lehet készíteni, többek között ezt használja például a Facebook és a Twitter a mély neurális hálózatok kutatására[9].

Használtam még a CUDA 8.0-t és a cuDNN v5.1-et, mely az Nvidia könyvtára a grafikus processzorok hatékony kihasználására neurális hálózatokkal. Ami a grafikus processzorokat illeti, két típussal is dolgoztam, melyeket a Budapesti Műszaki és Gazdaságtudományi Egyetem Távközlési és Médiainformatikai Tanszéke bocsájtott rendelkezésemre: Nvidia Titan X 12 GB VRAM-mal és Nvidia GTX 970 4GB VRAM-mal. Mindkét tanító szerver, melyekben ezek a grafikus processzorok voltak, Intel Core i7-es precesszorral és 32 GB RAM-mal rendelkezik.

[9] http://torch.ch/whoweare.html

5. Elvégzett munka

5.1. Adatfeldogozás

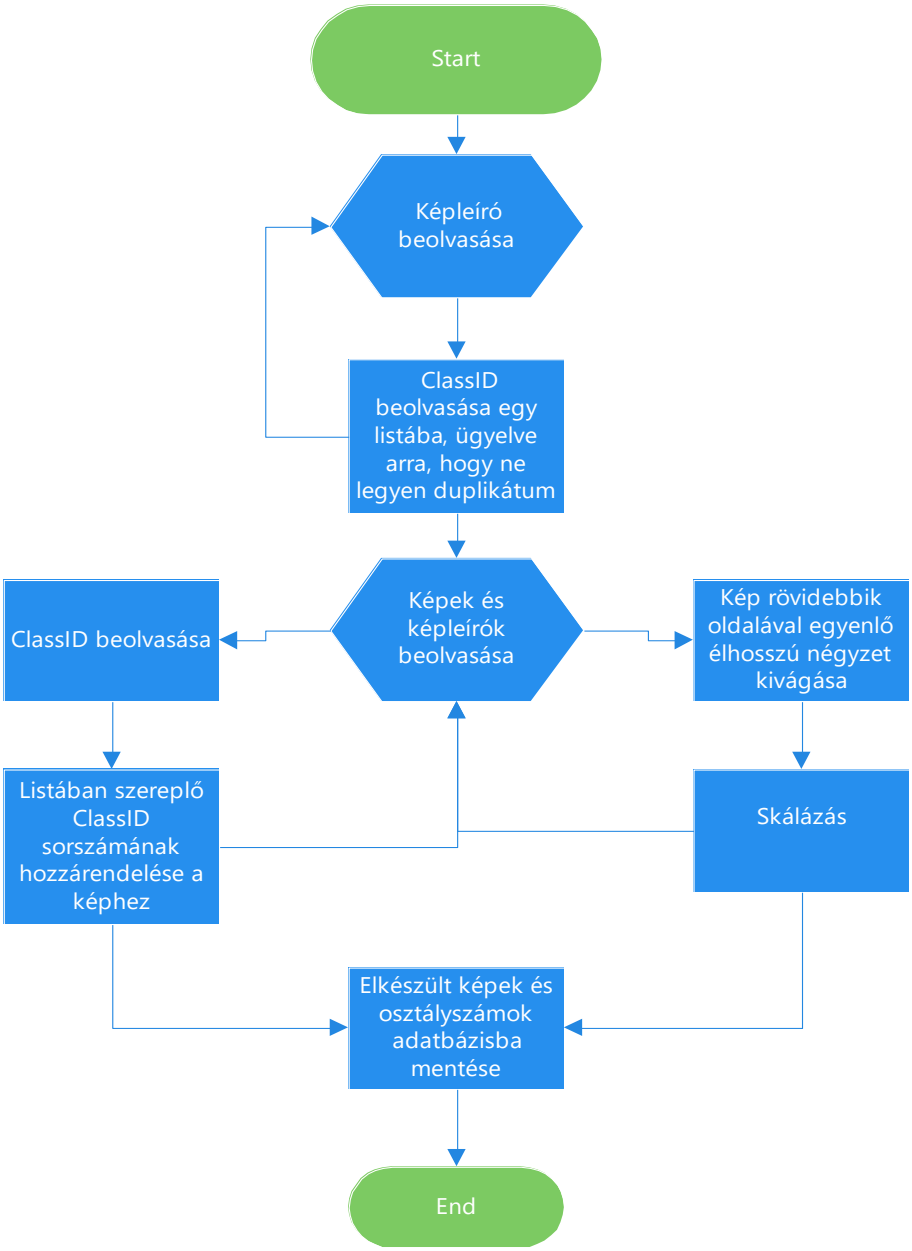
Először fel kellett dolgoznom a képeket és a hozzájuk tartozó leíró fájlokat. A képekhez tartozó leíró XML (Extensible Markup Language) fájlokban a számunkra érdekes ClassID egy 3-tól 5 jegyű véletlenszerűen generált szám. Ahhoz, hogy a hálózatot tanítani tudjam, ezekhez az ClassID-khoz 1-től 1000-ig kellett számokat rendelnem. Ezt úgy oldottam meg, hogy bejártam az összes leíró fájlt, és egy Lua listában kigyűjtöttem az összes ClassID-t. Fontos volt, hogy egy ClassID csak egyszer szerepeljen, úgyhogy minden listához adás előtt ellenőrizni kellett, hogy nincs e már a listában az éppen beolvasott ClassID.

Az elkészült listát az egyszerű átláthatóság miatt CSV (Coma-Separated Value) fájlba mentettem. A képek beolvasásakor az imént elkészített CSV fájlt beolvastattam egy listába. Az éppen beolvasott kép leírójában szereplő ClassID-t megkerestettem a listában, és az adott ClassID lista béli sorszámát rendeltem az éppen beolvasott képhez.

A beolvasott kép közepét kivágtam egy olyan négyzet alakban, melynek oldalhossza a kép rövidebbik dimenziójával egyezett meg. Majd kisebb méretűre skáláztam az adott képet (ennek nagysága hálózatspecifikus érték). Ezeket a képeket és a hozzájuk tartozó ClassID-kből készített lista sorszámokat Torch binárisba mentettem, úgy hogy 1 ilyen adathalmaz nem haladta meg az 1 GB-os méretet, így nem fogyott el a memória sem mentéskor sem beolvasáskor. A teljes folyamat a 16. ábrán figyelhető meg.

A datasetek beolvasása után a kiszámoltam az összes képre színcsatornánként a várható értéket és a szórást, majd ezeket átlagoltam. A minibatch elkészítésekor színcsatornánként normalizáltam a képeket 0 várható értékre és egységnyi szórásra (15. ábra). Ezt úgy tudtam megtenni, hogy a színcsatornánként kivontam a globálisan kiszámolt várható értéket és osztottam a globálisan kiszámolt szórással.

15. ábra: *Egy eredeti kép (balra) és egy előfeldolgozott kép (jobbra): közepének kivágása, nulla várható érték és egységnyi szórás beállítása után (skálázás nélkül)*



16. ábra: *Az előfeldolgozás folyamatábrája*

Ezután a tanitó adatok 10%-át kiválasztottam, ezek lettek a validációs adatok, és a hálózat ezekre adott hibáját figyeltem a túltanulás elkerülése végett. (A túltanulás esetemben azt jelenti, hogy konkrétan csak a tanító adatok osztályait tanulja meg a hálózat, azaz ezektől eltérő képre helytelen osztályt adna vissza.) Minden tanítási ciklus (epoch) után validáltam. Ha a validációs adatokra adott tévesztési mátrix pontossága növekedett egy lefutott epoch után, akkor elmentettem a hálózat aktuális állapotát.

A betanított modellek végső kiértékelésére pedig a 2015-ben kiadott besorolandó képeket használtam, ugyanis ezek kézzel címkézett osztályát a 2015-ös verseny után kiadták. Ezen hasonlítottam össze a hálózatok hatékonyságát különböző metrikákkal.

5.2. Tanítási alapelvek

Az összes tanítást ezen alapelvek szerint végeztem. A validációs adatokat pszeudo random módon választottam ki. Pszeudo random alatt azt értem, hogy megadtam egy inicializációs vektort a véletlen generátornak, így minden tanításhoz ugyanazokat a validációs adatokat választotta ki az algoritmus, ezzel objektív lett a tanítás, könnyen összehasonlíthatóvá váltak a különféle hálózatok eredményei.

A mini-batch-ekbe tartozó képeket mindig véletlenszerűen választottam ki. A tanítások alatt mindig 50-es türelmi értéket állítottam be, azaz ha 50 epoch után nem javult a tévesztési mátrix validációs pontossága, akkor megállt a tanítás, ez a korai megállás (angolul early stopping).

A tanítási és validációs szkripteket én készítettem el Lua nyelven. Költségfüggvénynek mindenhol ClassNLL-t használtam, mert ez volt a legmegfelelőbb egy ilyen klasszifikációs problémára.

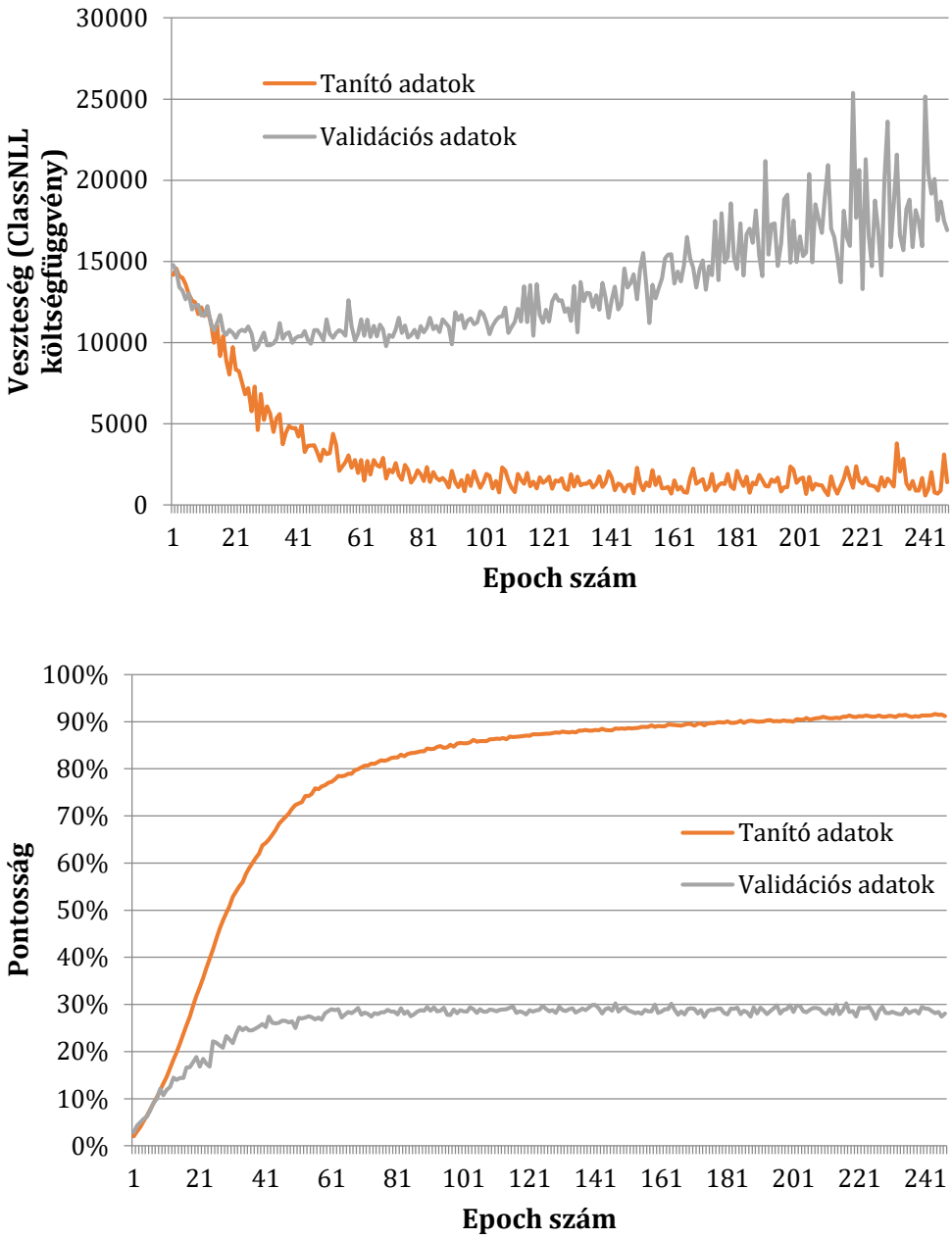
5.3. AlexNet

Először az AlexNet-et készítettem fel a probléma megoldására, hisz ez volt az első igazán hatékony konvolúciós neurális hálózat architechtúra. Továbbá rengeteg forrás állt rendelkezésre erről a hálózatról és ez egy sokféle területen sikerrel kipróbált háló. Ráadásul az erőforrásigénye is viszonylag kicsi, így viszonylag hamar megkaphattam a tanulás eredményét. Még batch méret optimalizálásra is jutott idő [1. táblázat]. Ezt a hálózatot alapjaitól tanítottam, nem használtam előretanított hálót. Ennél a hálózatnál Adadelta

optimializációt használtam, és ahogy látható a 16. ábra fenti részén, a tanulási hiba oszcillált emiatt. Az eredeti AlexNet architektúrát kiegészítettem BatchNormalization réteggel is, melytől azt vártam, hogy javítson a tanulási időn és csökkentse a tanulási hibát. Ehhez a hálózathoz 224x224 pixelesre skáláztam a bemeneti képeket, mert ekkora képekkel használható az AlexNet az első konvolúciós réteg dimenziói miatt. Végül a hálózatban szereplő ReLU aktivizációs függvényeket PReLU-kra cseréltem, ettől a pontosság javulását vártam.

Batch méret	**Tanítási pontosság [%]**	**Validációs pontosság [%]**	**Korai megállás #epoch**
20	96.19	32.98	189
30	89.57	29.26	138
40	93.96	26.54	199
50	92.99	25.32	209
60	68.35	24.48	124
80	89.33	23.52	226
100	87.79	31.63	102
130	90.28	37.76	148
150	84.5	20.66	286
200	47.31	20.96	208

1. táblázat: *Batch méret optimalizáció, a Tanítási pontosság a klasszifikáció tévesztési mátrixának helyes találatai a tanító adatokra, a Validációs pontosság pedig ugyan ez a validációs adatokra. A Korai megállás #epoch mutatja hányadik epoch-nál állt le a tanítás*



16. ábra: *AlexNet tanítása alatti veszteség alakulása (fent) és az átlagosan helyes sorok a tévesztési mátrixban (alul)*

5.4. Inception V2

A második architektúra, amit használtam, az AlexNet-nél jóval komplexebb Inception V2 volt. Először kerestem egy céljaimnak megfelelő Inception Torch7 implementációt, mely elő volt tanítva. Előtanításra általában az LSVRC versenyre kiadott adatbázist használják, ott

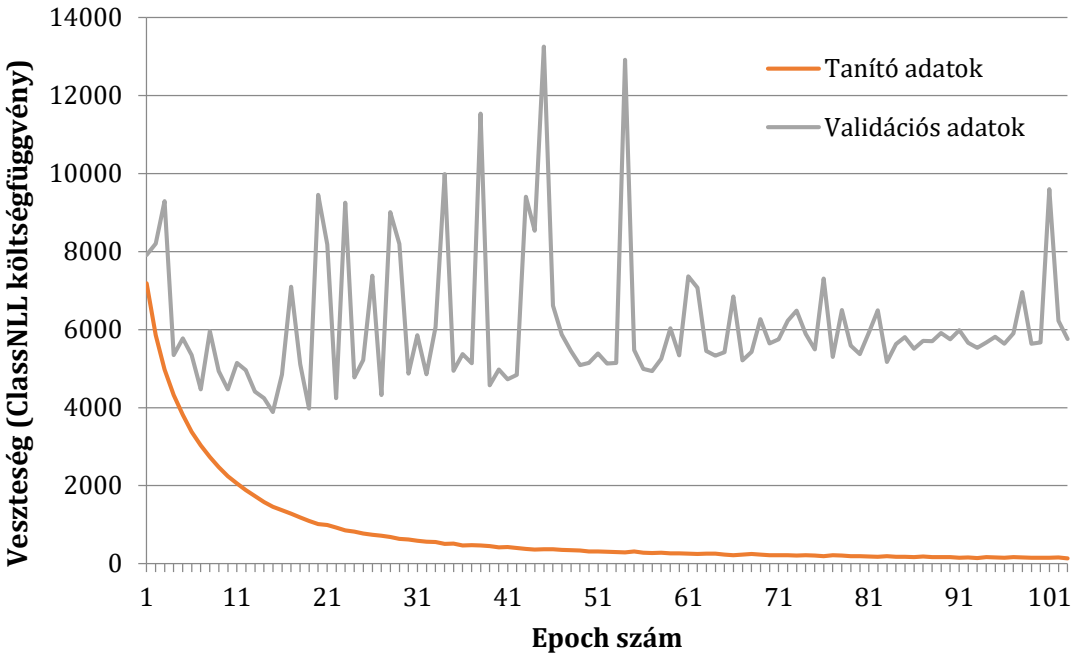
mintegy 2.2 millió kép van csoportosítva a kép tartalma szerint 1000 osztályba. Az LSVRC adatbázisban természetesen növények is megtalálhatók, de nem olyan részletesen címkézve, mint a PlantCLEF adatbázisában. Az előtanítás azért fontos, mert így a mélyebb konvolúciós rétegek rendkívül időigényes éldetekció szerű tanítását nem kell elvégezni. Annak oka, hogy az éldetekció szerű tanítás ilyen sok időt igényel az, hogy a hibavisszaterjesztő algoritmus a hibafüggvény gradiensét próbálja lekövetni, és mire az kezdeti réteghez ér, a gradiens nagyon lecsökkenhet, így csak nagyon kis változást (tanítást) tud előidézni. A továbbtanítás módszerét hívják a szakirodalomban *transfer learning*-nek, ilyenkor nem csak a klasszfikációs réteg, hanem a konvolúciós rétegek is tanulnak. Egy ilyen terjedelmű hálót körülbelül 2-3 hét lenne alapjaitól tanítani (egy olyan rendszeren, amit használtam), előretanított hálózat használatával ez a tanítási idő néhány napra csökkenthető. Továbbá a tudásból, amit a háló az LSVRC adatbázisból megtanult, sok minden hasznosítható a jelen tanításra, például az ég és a föld kinézete.

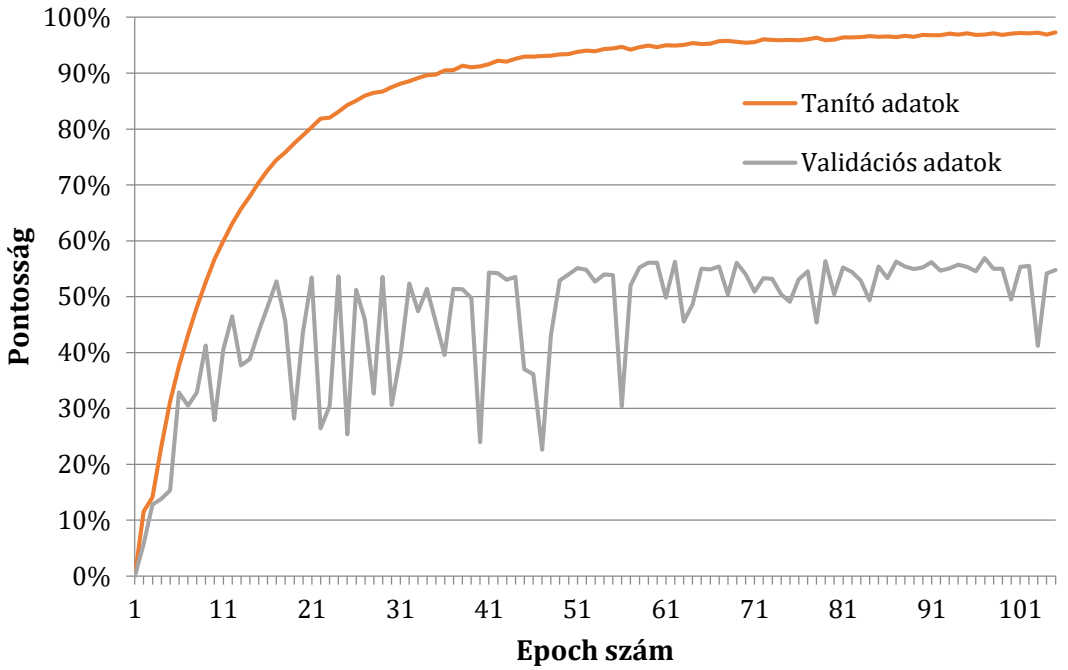
Ehhez a hálózathoz szintén 224x224 pixeles képekre volt szükség, ezért az adatfeldolgozásnál ekkora felbontású képeket készítettem skálázással. Az utolsó klasszifikációs réteg súlyait töröltem, ez a réteg egy fully connected layer, azaz minden neuron minden neuronnal össze van kapcsolva és ez végzi az osztályokba sorolást. E réteg súlyait azért inicializáltam újra, hogy a régi klasszifikációs osztályok helyét az új növény osztályok vegyék át. E művelet nélkül is betanítható lenne a hálózat, de így gyorsabbá válik a tanítás. Végül a hálózat legalján elhelyezkedő SoftMax réteget kicseréltem - az általam használt ClassNLL feltételes tanításhoz szükséges - LogSoftMax-ra, így az 1000 kimeneten megjelenő valószínűségek összege 1 lesz.

E háló tanítása kezdetben kudarcot vallott, mert Adadelta optimalizációval nem konvergált a hiba, azaz nem tanult. Az ok, amiért ez nem működött, több dologra is visszavezethető. Lehet, hogy az implementáció limitációja okozta, esetleg a Torch7-ben lehetett valami hiba vagy akár az Inception blokkok bonyolult architektúrája is eredményezhette, ugyanis a szakirodalomban nem találtam utalást az Inception és az Adadelta együttműködésére. A hibakeresés során az előtanított modell alapos vizsgálata után azt találtam, hogy az implementáció kódjában ki volt kommentezve egy Dropout réteg, ugyanis az az implementáció klasszifikációra volt tervezve, és ilyenkor nincs szükség

Dropout rétegre, mert ezzel rontanánk a klasszifikáció minőségét. Tanítás viszont csak ezzel a Droput réteggel volt megvalósítható.

Végül a Dropout réteg aktiválásával és SGD (Stochastic Gradient Descent) optimalizációval sikeresen tudott tanulni a hálózat.





17. ábra: *Inception V2 tanítása alatti veszteség alakulása (fent) és az átlagosan helyes sorok a tévesztési mátrixban (lent)*

5.5. Inception V3

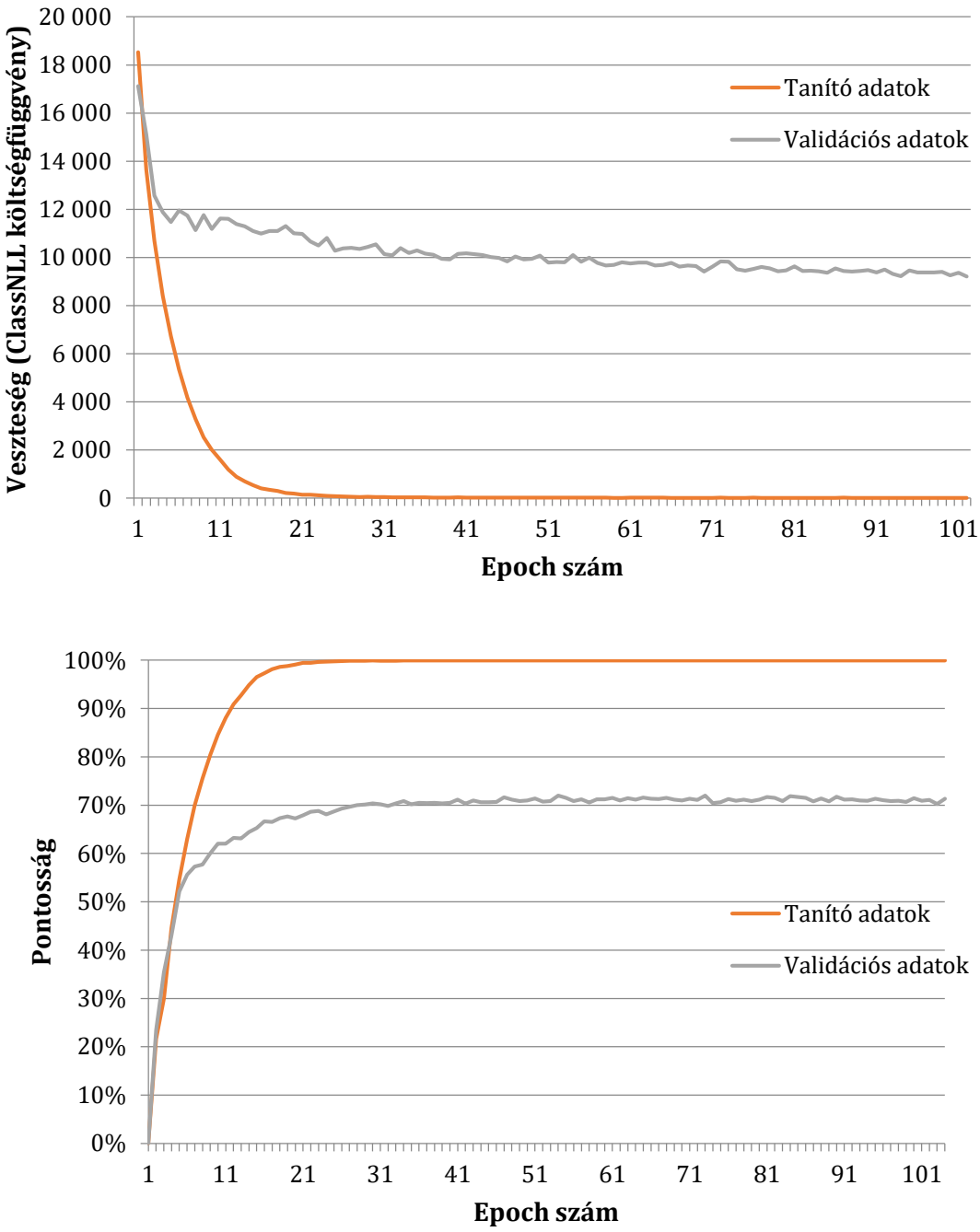
Miután az AlexNet és az Inception V2 működött, tovább szerettem volna növelni a pontosságot. A nemzetközi szakirodalmat áttekintve az Inception V3 hálózatot választottam. Különböző architektúrájú hálók implementációit is megvizsgáltam, de csak ennek a hálónak volt a kutatási célomhoz megfelelő mélységű implementációja.

Ez a hálózat még az Inception V2-nél és jóval összetettebb és több helyet is foglal a GPU memóriájában és tanítani is lassabb. Viszont az irodalom alapján ettől vártam a legjobb eredményeket.

Hasonlóképpen az Inception V2-höz, ez sem tanult semmilyen más optimalizációval, csak SGD-vel. Valószínüleg ismét az Inception blokkok komplexitása okozhatott gondot az Adadeltának. Nem tudta felmérni mekkora tanulási rátát állítson be, így a hiba nem csökkent, nagyon hamar oszcillálni kezdett. Ennél a hálónál is fontos volt, hogy előretanított Torch7 implementációt használjak, majd ezt a hálózatot tanítsam tovább. Az ILSVRC adatbázissal előretanított háló súlyainak letöltéséhez TensorFlow-ra volt szükség. Ez a Google által fejlesztett platform, mellyel Python alapon végezhető a neurális hálózatok tanítása. A súlyok letöltésére rendelkezésre állt egy Python szkript, majd egy Lua szkript, ami elkészítette az előretanított hálózatot (felépítette a hálót, majd beöltötte az imént letöltött súlyokat.)

E hálózat 299x299 pixeles képekre van tervezve, mert ezek több információt tartalmaznak, mint a 224x224 pixelesek. Ezért újra kellett generálnom a tanító adatokat.

Ennél a hálózatnál szintén újra inicializáltam az utolsó klasszifikációs réteg súlyait a gyorsabb tanulás érdekében. Továbbá, a klasszifikációs réteg utáni SoftMax réteget itt is kicseréltem a LogSoftMax-ra.



18. ábra: *Inception V3 tanítása alatti veszteség alakulása (fent) és az átlagosan helyes sorok a tévesztési mátrixban (lent)*

6. Kiértékelés

Munkám során körülbelül 1200 sor kód született, 25-30 tanítást futtattam le körülbelül 3 hét szerver kihasználás alatt.

6.1. AlexNet

Az AlexNet hálózat gyengén teljesített a másik két hálózathoz képest. Az 1. táblázat alapján, ez a háló 130-as batch mérettel teljesített a legjobban. Egy epoch körülbelül 5 perc alatt futott le, utána mindig következett egy validáció, ami körülbelül másfél percet vett igénybe. Így a teljes tanítás 148 epoch-ra körülbelül 16 óráig tartott a GTX 970 GPU-s szerveren. A tanításhoz Adadelta optimalizációt használtam, így nem kellett tanulási rátát beállítanom. A tanítás alatt a tévesztési mátrix legjobb validációs pontossága 37,76% volt. A legjobb validációs pontosság eléréséhez 10 tanításra volt szükség (batch méret optimalizáció). A 2015-ös teszt adatokra a top-1 pontosság, azaz azon képek száma, melyeket elsőre eltalált a háló 17,36%-ot, vagyis 3723 db kép a 21447 db-ból. A MAP-re (Mean Average Precision) első 10 találatra pedig 45,43%-ot kaptam.

A MAP (Mean Average Precision) metrikát egyszerűen úgy érdemes elképzelni, hogy ha rákeresünk egy növény osztályra, akkor az első adott számú találat között hány ténylegesen abba az osztályba tartozó növényt kapunk vissza. Minél előrébb helyezkedik el a találati listában az adott kép, annál nagyobb súllyal számít bele a végső értékbe. A 3. képlet az átlagos pontosságot (Average Precision) írja le “n” visszadott elem esetén. P(k) a k-adik elem pontossága, m a releváns osztályok száma, n pedig a megjósolt osztályok száma. MAP a 4. képlet alapján számolódik. A MAP számításhoz szükséges szkriptet én írtam Python nyelven.

$$ap@n = \sum_{k=1}^{n} \frac{P(k)}{\min(m,n)} \quad (3)$$

$$MAP@n = \sum_{i=1}^{N} \frac{ap@n_i}{\mathrm{N}} \quad (4).$$

6.2. Inception V2

Az Inception V2 háló lényegesen jobban teljesített az AlexNethez képest. Idő hiányában kevés optimalizációval 0.025-ös tanulási rátával, 45-ös batch mérettel tanítottam. Azért csak 45-ös batch mérettel, mert GTX 970 4 GB-os memóriájába nem fért el több. Ilyenkor nem csak a képeknek kell elférniük, hanem az egész hálózatnak és az összes hibavektornak. Egy epoch körülbelül 45 percig tartott, míg a validálás körülbelül 7 percet vett igénybe. Így a 105 epoch-os tanítás mintegy 4 napig tartott. A tanítás alatt a tévesztési mátrix legjobb validációs pontossága 56,89% volt. E validációs hiba elérése 3 tanítás alatt sikerült kisebb optimalizációkkal. A 2015-ös teszt adatokra a top-1 pontosság 46,41%. A MAP (mean-average-precision) első 10 találatra pedig 78,46%-ot kaptam.

6.3. Inception V3

Az Inception V3 háló teljesített a legjobban a három hálózat közül, viszont ez tanult a leglassabban a GTX 970 kártyán. Ennél a hálózatnál 0,035-ös tanulási rátát, 10^{-5}-es weight decay-t, 0,1-es momentumot és 10^{-7}-es learning rate decay-t használtam. Itt csak 12-es batch mérettel tanítottam, mert már csak ennyi fért el a memóriában a hálózat nagyobb komplexitása miatt. Egy epoch körülbelül 1 óráig tartott, a validálás pedig körülbelül 24 percig, így a 103 epoch lefutása és ezekhez tartozó validálás körülbelül 6 napig tartott. Ezt a hálózatot a próbáltam tanítani a TITAN X 12GB memóriájú videókártyán is, érdekes módon a tanítás nem volt számottevően gyorsabb, csak nagyobb batch mérettel is tudtam tanítani a több memória miatt, de ez végül nem javított az eredményen. A legsikeresebb tanítás alatt a tévesztési mátrix legjobb validációs pontossága 72% volt. E pontosság eléréshez 4 tanítás alatt jutottam a hiperparaméterek optimalizációjával. A 2015-ös teszt adatokra a top-1 pontosság 63,82%. A MAP-re (mean-average-precision) első 10 találatra pedig 88,61%-ot kaptam.

6.4. További próbálkozások

Az első újítási gondolat, ami felmerült bennem az volt, hogy a kép tartalma (a kép leíró fájlban content) alapján szét kellene válogatni a képeket, és külön hálókat betanítani ezekkel. Majd az azonos megfigyelésből származó többféle tartalommal rendelkező képekre adott kimeneteket súlyozni (például a levél alapján nagyon jó aránnyal lehet a növényt

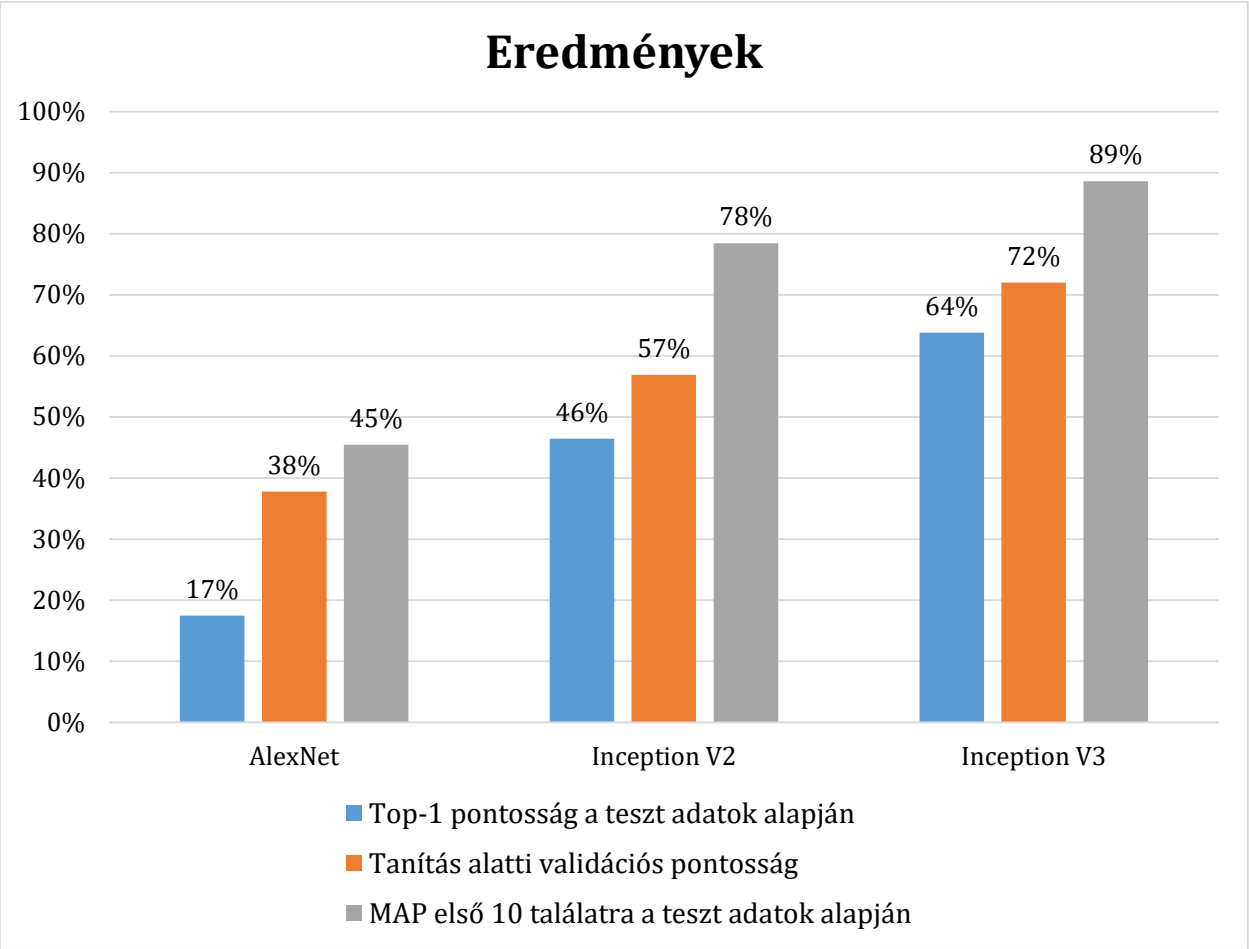
felismerni) és így meghatározni, hogy milyen növény van a képen. De ez a gondolat már más fejében is megfordult, sőt meg is próbálta a tavalyi versenyen, de érdekes módon nem vezetett eredményre [27].

A második hipotézisem az volt, hogy ha a növényeket Genus alapján szétválogatom, akkor azzal pontosabb modelleket fogok tudni létrehozni. Viszont Genus-ból rengeteg különböző volt, így inkább a Family-t, a növények családját használtam. Így lett 22 osztály, 21 különböző osztály, 1-be pedig ment a maradék: olyan családok, amelyben kevesebb, mint 10 faj volt. Ha ezt az egyszerűbb klasszifikációs feladatot sokkal jobban meg tudta volna oldani a háló, mint az egyedi fajok osztályozását, akkor a betanítottam volna még 22 hálót, amely már az elkülönített családokra tanul. Ezekből ismét kevesebb osztállyal már egyszerűbb lett volna a fajokat kitalálni. Viszont sajnos a 22 osztályra tanulás nem hozott olyan jó eredményt: Inception V3-al 72,11% volt a tévesztési mátrix validációs pontossága, a másik két hálózattal pedig ennél is gyengébb eredmények születtek.

További ötletem, mellyel növelhető lenne a klasszifikáció pontossága további tanító képek „generálása", azaz az adatok dúsítása a meglévő képek tükrözésével és forgatásával. Megoldás lehet még az ensemble modell, mely azt jelenti, hogy több azonos architektúrájú hálózatot betanítunk akár ugyanazokkal a tanító adatokkal, akár véletlenszerűen elosztott tanító adatokkal. Majd a kiértékelésnél többségi döntéssel, vagy akár súlyozással dönthetünk az értékelendő teszt kép osztályáról. A szakirodalom szerint az ilyen összetett modellek kismértekben ugyan, de javítanak a pontosságon.

6.5. Értékelés

Az eredmények (19. ábra) a várakozásiamnak megfelelően alakultak. Az Inception V3 háló tanult leglassabban (kb. 1 hetes tanítási idő), viszont ez érte el a legjobb eredményt (top-1 pontosság 63,82%, MAP első 10 találatra 88,61%). Az Inception V2 háló gyorsabban tanult (kb. 4 nap), de nem érte el az Inception V3 teljesítményét (top-1 pontosság 46,41%. A MAP első 10 találatra 78,46%). A legrégebbi háló, az AlexNet érte el a leggyengébb eredményt (top-1 pontosság 17.46%. A MAP első 10 találatra 45,43%), viszont ez az egyszerűsége miatt sokkal gyorsabban tanítható (kb. 8 óra).



19. ábra: *A végső eredmények*

7. Összefoglalás

A kutatómunkám célja az volt, hogy a PlantCLEF 2016 versenyen a rendelkezésre álló tanító növény-képek alapján minél jobb eredménnyel osztályokba soroljam a külön kiadott teszt képeket konvolúciós neurális hálózatok segítségével. Munkámból konferenciacikk is született [1], és folyamatban van a Multimedia Tools and Applications folyóirat cikk elbírálása szintén ebből a témából.

A szakirodalom áttekintése után kiválasztottam három Torch7 mély tanuló keretrendszer alatt készült, általános célokra előtanított konvolúciós hálózatot, az Inception V2-őt és az Inception V3-at. Továbbá egy nem előtanított hálót, az AlexNet-et. Először előfeldolgoztam a rendelkezésre álló tanító adatokat (levágás, skálázás, osztályok elrendezése), majd elkészítettem a tanító és validáló szkripteket. Megkerestem és beállítottam a megfelelő tanulási rátákat, batch méreteket és a további szükséges paramétereket. A hálózatokat ezután továbbtanítottam a cél képadatbázissal, mely eljárást angolul *transfer learning*-nek neveznek.

Miután a hálózatok elérték az adott hiperparaméterekkel a legjobb eredményeiket (a tévesztési mátrix legjobb validációs pontosságát), leteszteltem a hálózatokat a 2015-ben kiadott teszt adatokkal. Majd az eredményeket többféle metrikával (Top-1 pontosság, MAP) értékeltem ki.

Ahogy a teszt eredményei mutatják, az általam tanulmányozott és alkalmazott konvolúciós hálózatok a feladatra szabva alkalmasak ilyen típusú klasszifikációra. A paraméterek további „finomhangolásával" még jobb eredményeket lehetne elérni, hiszen ahhoz, hogy egy feladatot jól modellezzünk, legalább 100 tanítás szükséges.

A szakirodalom alapján kialakult várakozásaim teljesültek, a legjobb eredményt elérő hálózat az Inception V3 lett, míg a leggyengébb az AlexNet.

A legjobban teljesítő Inception V3 hálózat a teszt eredmények alapján 64%-os pontossággal képes megmondani a kép alapján a növény pontos fajtáját a betanított 1000 osztályból, ezzel eldöntve, hogy a növény kártékony e vagy sem. Sajnos nem állt rendelkezésemre olyan teszthalmaz, melyben külön voltak válogatva a kártékony növények, de ebből a 64%-os pontosságból következtethetünk arra, hogy ennél jóval nagyobb valószínűséggel képes megállapítani a hálózat automatikusan, hogy a növény kártékony-e a fénykép alapján.

8. Irodalomjegyzék

[1] Tóth, B. P., Papp, D., Osváth, M. & Szűcs, G. (2016) "Deep learning and svm classification for plant recognition in content-based large scale image retrieval."Working notes of CLEF 2016 Conference. 2016.

[2] Berg, A., Deng, J., & Fei-Fei, L. (2010) Large scale visual recognition challenge http://www. imagenet. org/challenges.

[3] McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4), 115-133.

[4] Hebb, D. O. (1949). The organization of behavior: A neuropsychological theory.New York, 4.

[5] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6), 386.

[6] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1988). Learning representations by back-propagating errors. Cognitive modeling, 5(3), 1.

[7] Nickolls, J., & Dally, W. J. (2010). The GPU computing era. Micro, IEEE, 30(2), 56-69.

[8] Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In Proceedings of COMPSTAT'2010 (pp. 177-186). Physica-Verlag HD.

[9] Qian, N. (1999). On the momentum term in gradient descent learning algorithms. Neural networks, 12(1), 145-151.

[10] Moody, J., Hanson, S., Krogh, A., & Hertz, J. A. (1995). A simple weight decay can improve generalization. Advances in neural information processing systems, 4, 950-957.

[11] Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701.

[12] Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th International Conference on Machine Learning (ICML-10) 807-814.

[13] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE International Conference on Computer Vision 1026-1034.

[14] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1), 1929-1958.

[15] Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

[16] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[17] LeCun, Y., Jackel, L. D., Bottou, L., Cortes, C., Denker, J. S., Drucker, H., ... & Vapnik, V. (1995). Learning algorithms for classification: A comparison on handwritten digit recognition. Neural networks: the statistical mechanics perspective, 261, 276.

[18] http://image-net.org/

[19] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems 1097-1105.

[20] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 1-9

[21] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015). Rethinking the inception architecture for computer vision. arXiv preprint arXiv:1512.00567

[22] Wang, M., Liu, B., & Foroosh, H. (2016). Factorized Convolutional Neural Networks. arXiv preprint arXiv:1608.04337.

[23] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385.

[24] Chollet, F. (2016). Deep Learning with Separable Convolutions. arXiv preprint arXiv:1610.02357.

[25] Göeau, H., Joly, A., Pierre, B. (2015): LifeCLEF Plant Identification Task 2015. CLEF working notes 2015

[26] http://torch.ch

[27] Choi, Sungbin. "Plant identification with deep convolutional neural network: Snumedinfo at lifeclef plant identification task 2015." Working notes of CLEF 2015 conference. 2015. [28] Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaptation. Neural networks, 1(4), 295-307.

[29] Baldi, P., Brunak, S., Chauvin, Y., Andersen, C. A., & Nielsen, H. (2000). Assessing the accuracy of prediction algorithms for classification: an overview. Bioinformatics, 16(5), 412-424.

Köszönetnyilvánítás

Elsősorban szeretnék köszönetet mondani a konzulensemnek, Dr. Tóth Bálint Pálnak áldozatos munkájáért és segítségért. Továbbá köszönöm a Budapesti Műszaki és Gazdaságtudományi Egyetem tanszékének a Távközlési és Médiainformatikai Tanszéknek, hogy rendelkezésemre bocsátották a tanításokhoz szükséges szervert.

Köszönettel tartozom még családomnak: páromnak, Patríciának és kislányomnak, Zoénak támogatásukért.