Neurális hálózatok vizsgálata képosztályozás feladatára

Sipos Levente - NLLIEC

2023.06.02

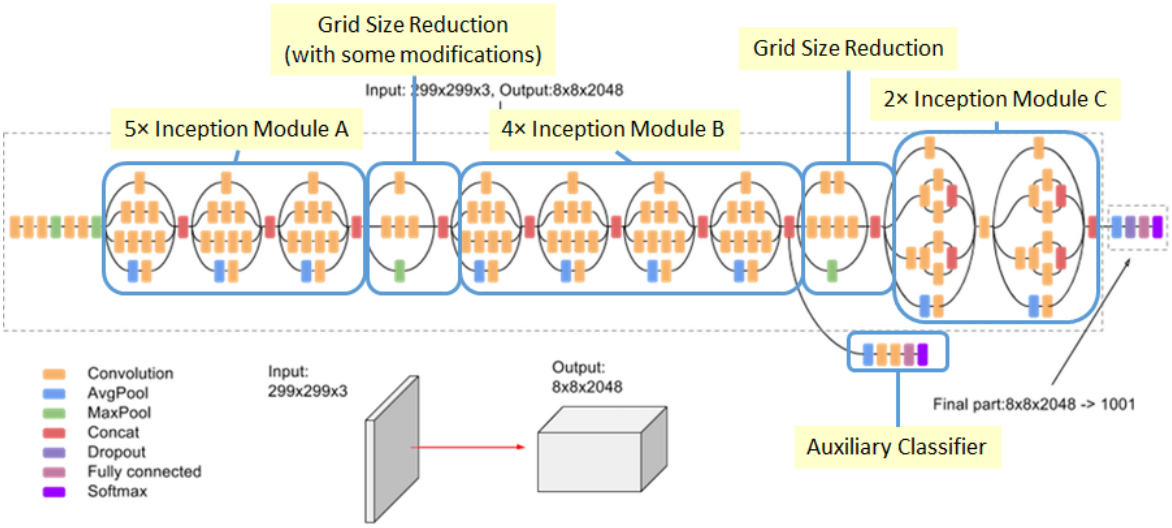
I. Bevezetés

Önálló laborom témájaként mély konvolúciós neurális hálózatok vizsgáltam a CIFAR-100-as adathalmazon. Ez egy gyakran használt adathalmaz a mélytanulás területén. 100 különböző osztályba sorolt összesen 60 000 színes képből áll, amelyek 32x32 pixel méretűek. Minden osztályba 600 kép tartozik, és az adathalmaz 50 000 - 10 000 arányban van bontva tanító és teszt halmazokra. Az adathalmaz hosszútávú népszerűségét az okozza, hogy a képek objektumát széles választékát fedik le, sok különböző szögből, így a valós világban előforduló sokszínű képek összességét adja. Emiatt tökéletes képosztályozási feladatra.

A félév alatt felépítettem saját hálózatokat, többnyire a népszerű InceptionV4-es architektúrából kiindulva, többféle változtatást megvizsgálva a modell több változatán és transfer learning alkalmazásával tovább tanítottam a Keras keretrendszerbe beépített InceptionV3 modellt.

Az Inception architektúrát a Google Research csapata mutatta be 2014-ben és azóta több fejlettebb változatát is közzé tették. Az architektúra fontos része az "Inception modul", amely egy építőelem, ami különböző szűrőméretek párhuzamos konvolúcióját hajtja végre, majd az eredményeket konkatenálja. Ez lehetővé teszi a hálózatnak, hogy különböző szintű jellemzőket tanuljon meg a tanító adathalmazon. Az Inception V1, az architektúra eredeti megvalósítása volt. Több egymásra rakott Inception modult tartalmazott, amelyek 1x1, 3x3 és 5x5 konvolúciókat, valamint pooling műveleteket használtak. Az InceptionV2-ben bevezették a Batch Normalizációt, ami normalizálja a konvolúciós rétegek bemenetét, ezzel elkerülve az „internal covariate shift” problémát. A normalizáció előnye még, hogy a nemlinearitás réteg normalizált adatokat kap, ami segít elkerülni a vanishing gradient és exploding gradient problémákat. A Batch Normalization hatékonyabb és stabilabb tanítást tett lehetővé, valamint javította a háló általánosító képességét is, és ma már szinte elengedhetetlen bármilyen mély konvolúciós háló tanításánál. Az InceptionV3 további javításokat hozott az architektúrába, mint például a nagyobb

konvolúciók faktorizációja több, kisebb konvolúcióvá és a label smoothing. Az InceptionV4-be beépítették a ResNet architektúra koncepcióit és ötleteit. A reziduális összekötések lehetővé teszik a gradiensek mély rétegekbe való eljuttatását, így a mélyebb rétegben nem vesznek el olyan információk melyeket a háló már tud csak a mélyebb rétegeknek nincs hozzáférésük. A felépített modelleken vizsgáltam az adat augmentáció hatását, az adatmennyiség befolyását, valamint egyéb teljesítmény metrikákat.



1. Az InceptionV3 architektúrája.

Mind a saját hálóm megépítésére, mind a transfer learninghez a Keras keretrendszert használtam, ami a Tensorflow keretrendszerre épül, és könnyen használható eszközöket ad a mély neurális hálók tanításának minden lépéséhez. A modellek kiértékelése és egyéb számítások során olyan gyakran használt könyvtárakat használtam, mint a numpy és a scikit-learn.

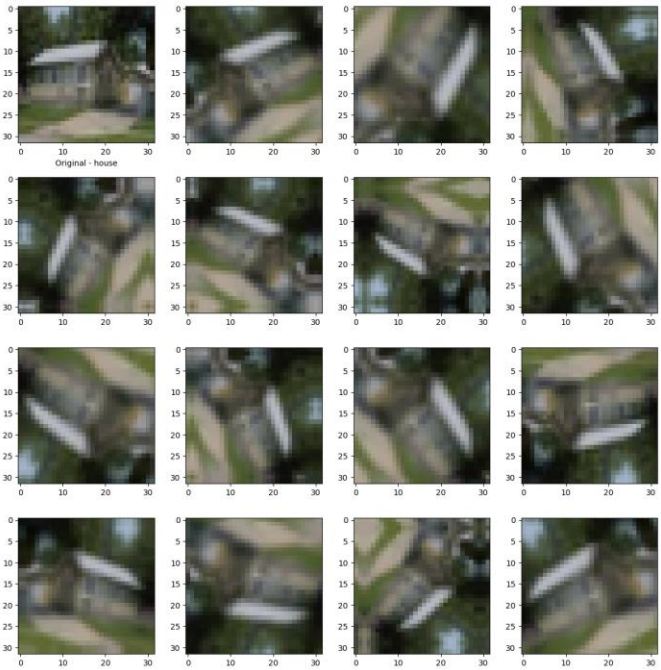
II. Adatok előfeldolgozása

Ahhoz, hogy egy neurális hálózat hatékonyan, vagy akár egyáltalán tudjon tanulni, az adatokat megfelelő formátumba kell hozni. A képeket a megszokott 0 - 255-ig terjedő skáláról, az Inception hálók tanítása során is alkalmazott -1 - +1 tartományba transzformáltam. Az adathalmaz tisztítására szerencsére nincs szükség, mivel a CIFAR-100 egy jól összeválogatott adathalmaz, nem tartalmaz rosszul felcímkézett vagy rossz minőségű képeket.

Az adatokon többféle augmentációt is végeztem. A Keras előfeldologozási rétegei közül használtam a Rescaling, Resizing, RandomFlip, RandomRotation és RandomZoom rétegeket. Ezen rétegek használatának előnye, hogy beleköthetők a háló modelljébe, mintha például konvolúciós rétegek lennének és így a GPU-n történnek a számítások, ami gyorsabb mintha

külön végeznénk el az előfeldolgozást CPU-n, mert így a batchekre egyszerre történik meg a transzformáció.

A Rescaling réteg valósítja meg a képek -1 - +1 tartományba transzformálását. Ez egy olyan lépés, amit minden képre el kell végezni, mind a tanítás, mind a predikció során. Ebből adódott is problémám, mert amikor augmentált adatokra vizsgáltam a hálót válaszát, ezeken a rétegeken átküldve a képeket, a Rescaling réteg hatása kétszer hajtódott végre, nem csak egyszer, mint a többi réteg, amik predikció során inaktívak. A Resizing réteget, csak transfer learning-nél kellett használnom, mivel csak ott kellett, hogy a képek 299x299 pixel méretűek legyenek. A RandomFlip réteg tükrözi a képet az x, y vagy mindkét tengelyre. A RandomRotation réteg a megadott mennyiségen belül forgatja a képet órajárással megegyező vagy ellentétes irányban, míg a RandomZoom réteg a megadott mennyiségig ránagyít a képre. Ezen rétegek egymás után kötésével a meglévő adatokat nagyon diverz módon transzformálhatjuk, amivel a háló általánosítóképességének javulását várjuk. Viszont a rétegek véletlenszerű mértékben fejtik ki hatásukat a bemenetükre, így amikor az adathalmazt többszörösen összekötve végeztem a hálók tanítását, egy ekkora adathalmaznál nem garantált, hogy nem keletkezik két ugyanúgy transzformált kép, de ez szerencsére nem jelent akadályt a háló számára.



2. Példa az augmentáció eredményére

Az adatokat szokásos 3 részre osztottam, tanító validációs és teszt adathalmazra. A Kerasból betöltött CIFAR-100-as adathalmaz már magában külön tartalmazza a tanító és tesz adathalmazt, így a tanító adathalmaz 20%-át szántam validációs adathalmaznak. Ez egy eléggé szokásos méret, ha rendelkezésre áll elég tanítóadat, és a tanítások során nem indokolta semmi, hogy ezen változtassak.

III. Az általam felépített háló

Az általam épített háló az InceptionV4 architektúra, reziduális összekötéseket nem tartalmazó változatán alapul. Jelentős különbség, hogy az Inception architektúra 299x299 pixeles képekre lett tervezve, míg az enyém a CIFAR-100-as adathalmaz 32x32-es képméretére. Ezt a döntést nem csak a képek méretbeli különbsége okozta, mert a képek méretét interpolációval meg lehet növelni a kívánt 299x299-esre – amire a transfer learninghez szükség is volt – hanem elsősorban a modellek mérete közti nagyságrendbeli különbség. Az én modelljeim pár millió paraméterrel rendelkeznek, míg az InceptionV3 és V4 24 millió, illetve 42 millió paraméterrel rendelkeznek. Ez teszi a különbséget az 1-4 és a 4-10 órás tanítások közt. A kisebb modell méretnek köszönhetően jóval több tanítást tudtam elvégezni és több mindent kipróbálni.

A háló legalapvetőbb építőeleme a Conv2DBatchNorm réteg, ami egy sima konvolúciós réteg után a Batch Normalizációs és aktivációs rétegekből áll. Az egész hálóban ezt az „új réteget” használom sima konvolúciók helyett.

A háló további nagyobb egységei az Inception-A, Inception-B, Inception-C, Reduction-A, Reduction-B modulok és a Stem. A Stem egyfajta előfeldolgozását végezi a bemenetnek. Csökkenti a térbeli dimenziókat, valamint növeli a jellemzők számát és összetettségét, ezáltal elősegíti a hálózat alacsonyabb szintű jellemzőinek felismerését. Itt az eredeti InceptionV4 a bemeneti kép méretét körülbelül negyedére csökkenti, ami a 32x32-es képeknél egy túl nagy méretbeli csökkenést jelentene, amivel a későbbi rétegek már nem tudnak jól dolgozni, így itt úgy döntöttem, csak az eredeti kép méretének körülbelül feléig csökkentem a méretet. Így az Inception-A blokk 13x13-as, 192 csatornás bemenetet kap.

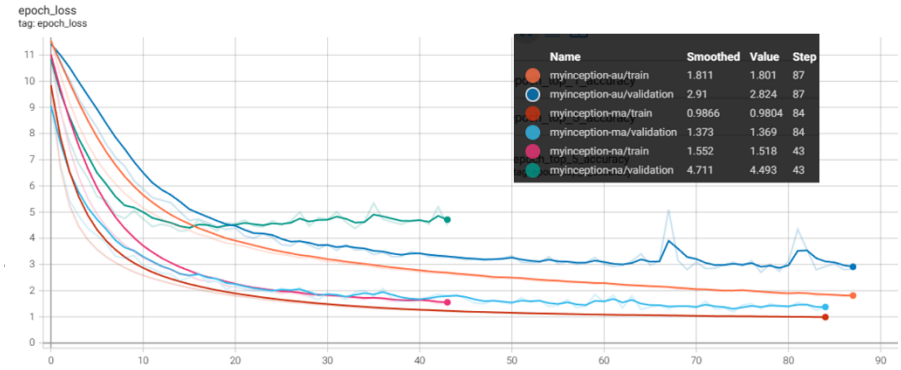
Az Inception blokkok feladata a bemeneti jellemzőtér transzformálása a bemeneti méret és csatornaszám meghagyásával. Ezt konvolúciós rétegek több párhuzamos ágon egymás utáni alkalmazásával, majd a blokk végén konkatenálja az ágakon kapott csatornákat. A különböző Inception blokkok a képek különböző szintű jellemzőit tanulják meg. A bemenet közelebbi blokkok alacsony szintű jellemzőket tanulnak meg, mint például élek, sarkok vagy egyszerű görbületek. A későbbi rétegek egyre komplexebb, a korábbi jellemzőkből felépített új jellemzőket tanulnak meg.

A Reduction-A és Reduction-B blokkok felelnek a bemenetek méretének csökkentéséért. Míg az Inception blokkok nem változtatják sem a méretet, sem a csatornák számát, csak átalakítják azokat, addig a Reduction blokkok csökkentik a bemenet méretét és növelik a csatornák számát, ezzel előkészítve a bemenetet a következő Inception blokknak.

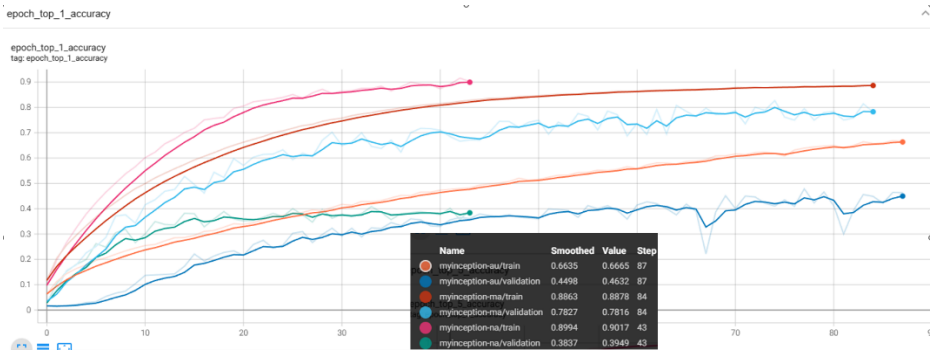
Az eddig felsorolt elemei a hálónak végzik a jellemzők kinyerését. Ezután a háló legvégén van egy Pooling, Flatten és Dense réteg, ami végső soron a megtanult jellemzők alapján valamelyik osztályba sorolja a képet.

A hálózat felépítésénél próbáltam megtartani az InceptionV4 architektúráját az Inception blokkokban is. A változtatások elsősorban a konvolúciós filterek számában jelennek meg. Mivel

az általam felépített modellt sokkal kisebbnek terveztem így a tanítható paraméterek számának csökkentését elsősorban a konvolúciós filterek számának csökkenésével értem el. Ez nem csak a modell paraméterszámának csökkentésére, és ezzel a tanítási idő csökkentésére hasznos, hanem alapvetően nincs is szükség ennyi paraméterre. Egy 32x32-es kép a CIFAR-100-ból nem tartalmaz annyi információt, mint egy 299x299-es kép az Imagenet adathalmazból, hiszen 100-ad akkora a kép és így sok kisebb és nagyobb részletet nem lesz képes a háló megtanulni, mert egyszerűen nincs mit megtanulnia. Ellenben a felesleges paraméterek, nem csak a tanítást lassítják, hanem nagyon megnehezítik a háló konvergálását vagy akár teljesen ellehetetlenítik a tanulást. Többféle filterszám csökkentést is kipróbáltam, az eredeti megtartásától kezdve a filterek nagyrészének eldobásáig. A túlságosan nagy filter szám általában lassú tanításokban, lassú konvergálásban végződött. Mindezek akkor jelentettek problémát amikor a háló hosszabb ideig tanult és el is ért eredményeket, ugyanis sokszor hamar beleesett lokális minimumokba, ahonnan nem tudott kitalálni. A túl kicsi filter szám hasonlóan eredménytelen tanításokban végződött. Habár epochonként gyorsabb volt a tanítás, a hiba és precizitás hamar elkezdett konvergálni és a javulás mértéke onnantól nagyon minimális volt. Végül a tapasztalatok alapján az egész hálóban megfeleztem a filterek számát, így egy 5.5 millió paraméteres hálót kapva. Minden modellt háromszor tanítottam, egyszer az alap 50 000 képen, egyszer augmentálva és egyszer 200 000 képpel, augmentálva. Az alapvető modell eredményei a következők:



3. A tanítások hibája epochonként



4. A tanítások Top-1 pontossága.

5. A tanítások Top-5 pontossága

Az elnevezések feloldása:

MyInception – A modell neve

NA – Nincs augmentáció

AU – Augmentáció 50 000 képen

MA – Augmentáció 200 000 képen

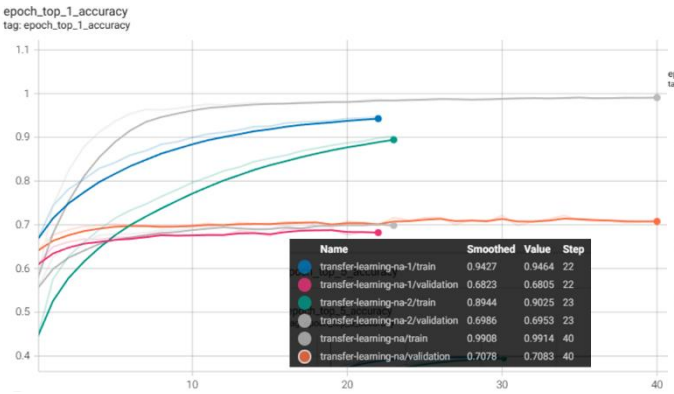
A fenti ábrákon nagyon jól látható, hogy míg az augmentáció nélküli tanítás, gyorsan konvergál 99%-os pontossághoz a tanítóhalmazon, addig a validációs halmazon vett pontossága leragad 68%-nál. Az augmentált adatokon tanított modell, tovább képes tanulni, és jobb eredményeket ér el a validációs halmazon. A még több adaton tanított háló, szinte teljesen megszűntette a tanító és validációs halmaz közti diszparitást. Ez mutatja, hogy bár sok adattal tanítottuk a hálót, ami alapvetően szükséges bármilyen jó eredmény eléréséhez, ha adat augmentációval (vagy egyéb módokon) még több adathoz tudunk hozzájutni, az csak segíti a háló tanulását és javítja az általánosítóképességét. Overfitting legjobban az augmentáció nélküli tanítás során jelentkezett, a többi esetben mérsékeltebb volt, és a tanítások elsősorban azért álltak le mert már nem javultak tovább és nem azért, mert a modell túltanult a tanítóadatokra.

Ezután az eredmények után, még tovább próbálkoztam más méretű hálók tanításával is. Megvizsgáltam másik optimalizációs eljárás hatékonyságát, de a mélytanulásban már szinte megszokott módon az Adam volt a legjobb, bár az RMSProp, amivel eredetileg tanították az Inception hálókat (amikor az Adam még egyébként nem létezett) sem teljesített sokkal rosszabbul. Ez nem meglepő, hiszen az Adam egy „továbbfejlesztett” változata az RMSProp eljárásnak. Emellett vizsgáltam még a háló végén lévő osztályozó változtatását például új rétegek beszúrásával, nagyobb Dropout értékkel, de ezek is hasonlóan csekély különbséget mutattak. A konvolúciós rétegekben végrehajtott változtatásoknál már nagyobb eltérést láttam, amikor több konvolúciós réteg méretét is változtattam, vagy az InceptionV3-hoz hasonló módon fasktorizáltam. A változások nagyobbak voltak, de még mindig nem voltak elég jelentősek ahhoz, hogy külön kezeljem őket. Az ilyen modelleket nem mentettem el, mivel nem tartottam szükségesnek.

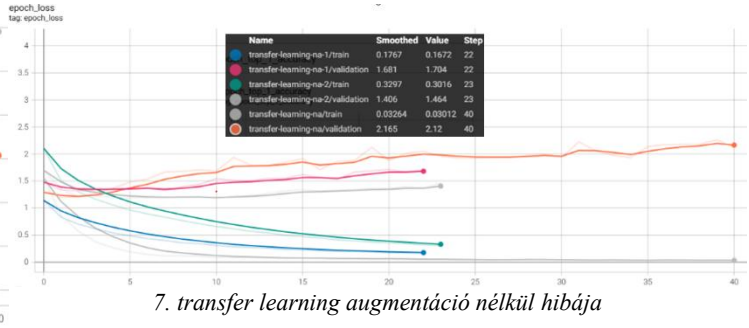
IV. Transfer Learning

A modellemet az InceptionV4-es architektúrára alapoztam, de a transfer learninget az Imagenet-en tanított InceptionV3-on végeztem. Ennek oka, hogy ezt a modellt és a megtanult súlyokat könnyen be tudtam tölteni Kerasból. Az architektúrák közt van különbség, de az általam felépített háló is eltér annyiban az InceptionV4-től, hogy ahhoz hasonlítva sem lehet pontos összehasonlítást végezni. A transfer learningnek nem is ez volt a célja, hanem hogy megvizsgáljam, hogy egy nagy adathalmazon betanított háló, milyen eredményt tud elérni az ott megtanult jellemzőkkel a CIFAR-100-on és viszonyítási alapot adjon az én hálóm által elért eredményhez.

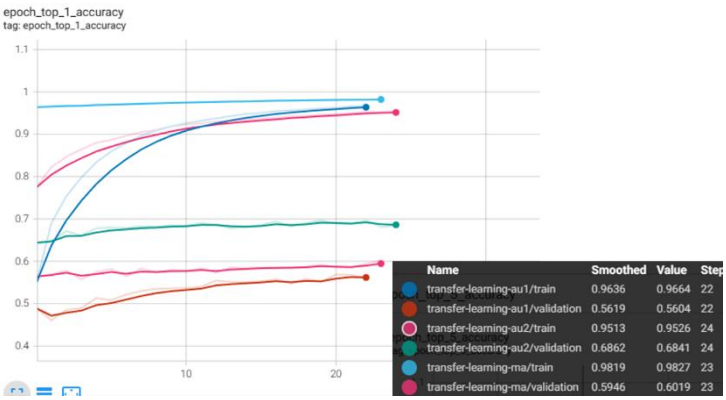
Ahhoz, hogy a 32x32-es képekkel lehessen tovább tanítani a hálót a Resizing réteg segítségével átméreteztem őket. A háló ezután hasonló módon a már felsorolt augmentációs rétegekkel és három adathalmazzal tanítottam. Először csak a kimenethez legközelebbi két Inception blokkot és Dense réteget tanítottam, majd néhány epoch után még több réteget taníthatóvá tettem és így folytattam a tanítást. Itt jelentősen kevesebb tanítást végeztem, mivel az InceptionV3 modell 24 millió paraméterét jelentősen több idő, így a tanítások puszta időbeli igénye is kihívást jelentett. A következő képeken látható, a legsikeresebb transfer learning tanítások eredménye:



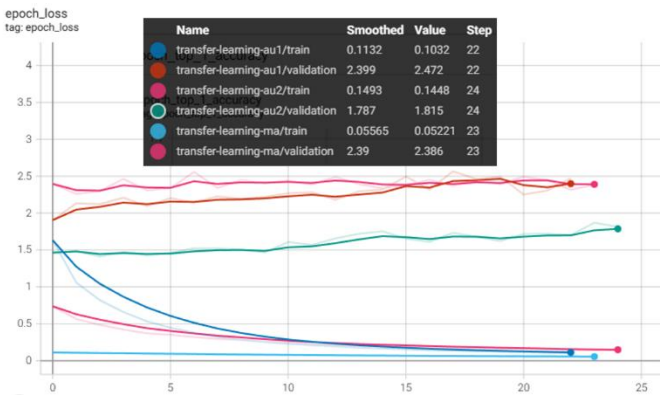
6. transfer learning augmentáció nélkül Top-1 pontossága



7. transfer learning augmentáció nélkül hibája



8. Transfer Learning augmentációval Top-1 pontosságai

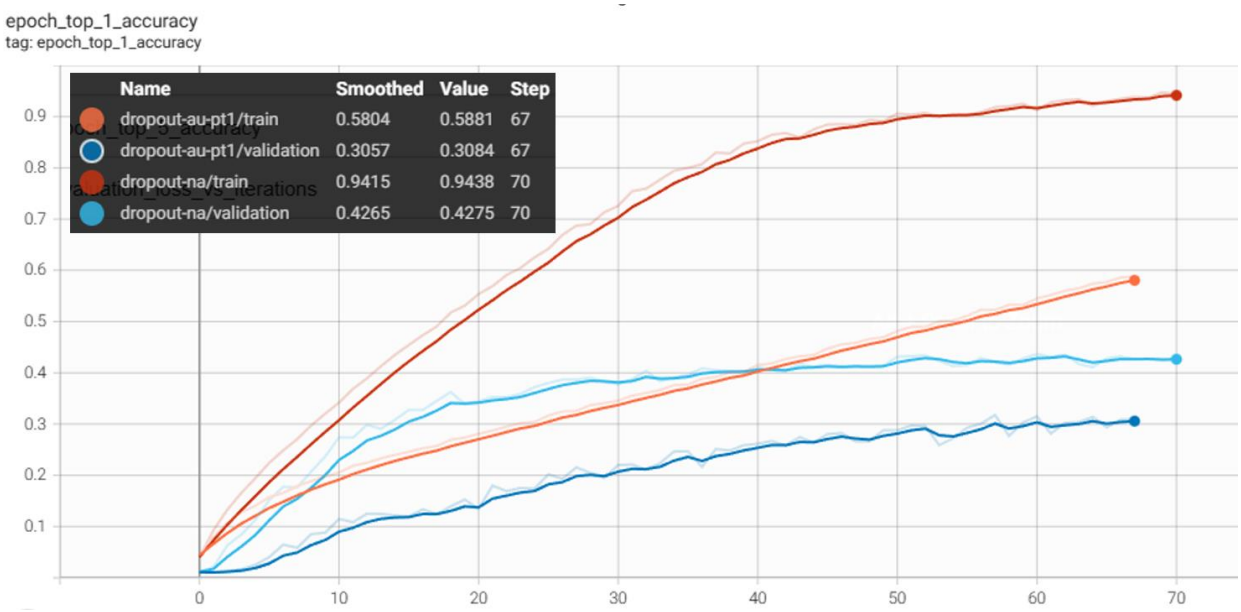


9. Transfer Learning augmentációval hibái

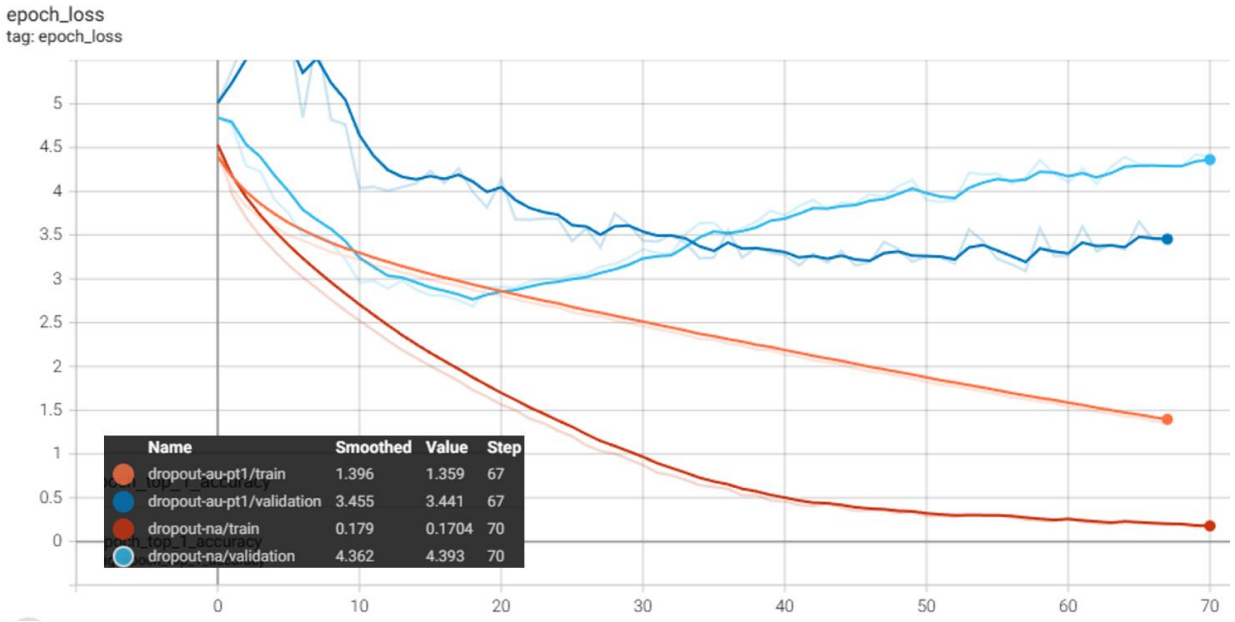
Mindegyik tanítás nagyon hasonló eredményeket ér el. Ami az ábrákon kitűnik, hogy volt egy olyan tanítás, ahol a háló nem volt képes kijutni a 50%-os Top-1 pontosság körül, és hasonlóan lapos, lassan javuló, csak éppen fele olyan jó eredményeket adott. Ennek magyarázata lehet egy nagyon szerencsétlen lokális minimumba való beragadás, bár ennek valószínűsége elég kicsi, ami valószínűbb, hogy a tanítás során elrontottam valamit és nem vettem észre. Ettől függetlenül a tovább tanított InceptionV3 háló várhatóan nagyon jó eredményeket adott. A tanítás alatt kis mértékű javulás is várható eredmény, hiszen az Imageneten már megtanult jellemzőkön nem kell és nem is nagyon lehet sokat javítani, mert azok alapvetően kiválóak.

V. Dropout alkalmazása

A Dropout egy gyakran használt technika, mely véletlen valószínűséggel kinullázza egy adott neuron értékét. A kinullázott neuronokkal lényegében különböző részhálókat hozunk létre a tanítás során, melyek szakértői együttest alkotnak. Neuronok helyett a konvolúciós hálókban neuronoknak megfelelő csatornákra alkalmaztam dropoutot. Minden Inception és Reduction blokk végén, a csatornák konkatenálása után alkalmazzuk a dropoutot, így eldobunk néhány csatornát és hasonlóan réshálókhoz jutunk. Itt kisebb, 0.3-as, dropoutot alkalmaztam, mint MLP-ben szokás (0.5-0.8), mert egy kieső csatornának nagyobb a hatása. Az így kapott eredmények:

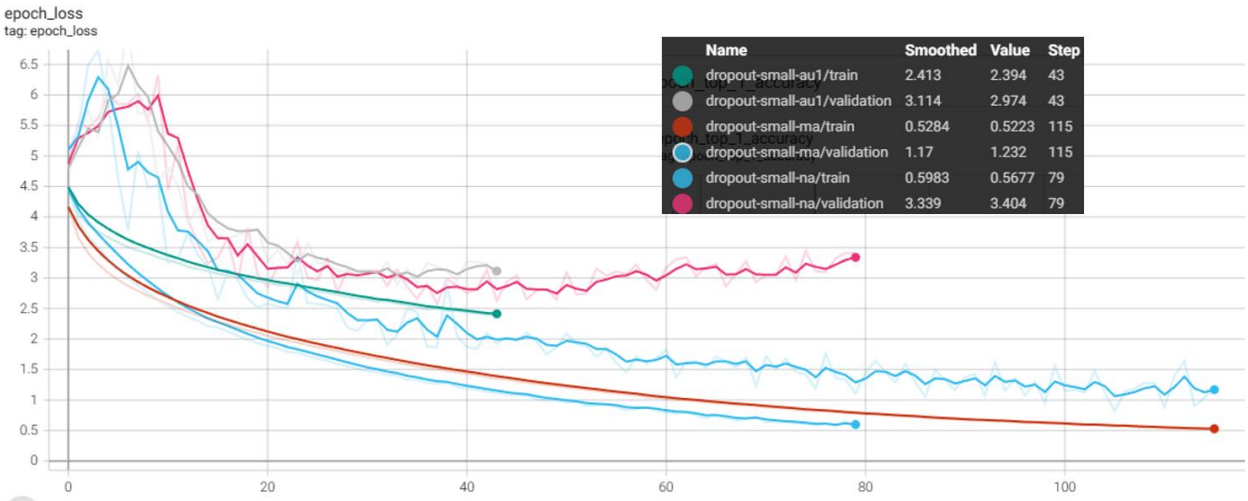


10.Top-1 pontosságok dropout alkalmazásával

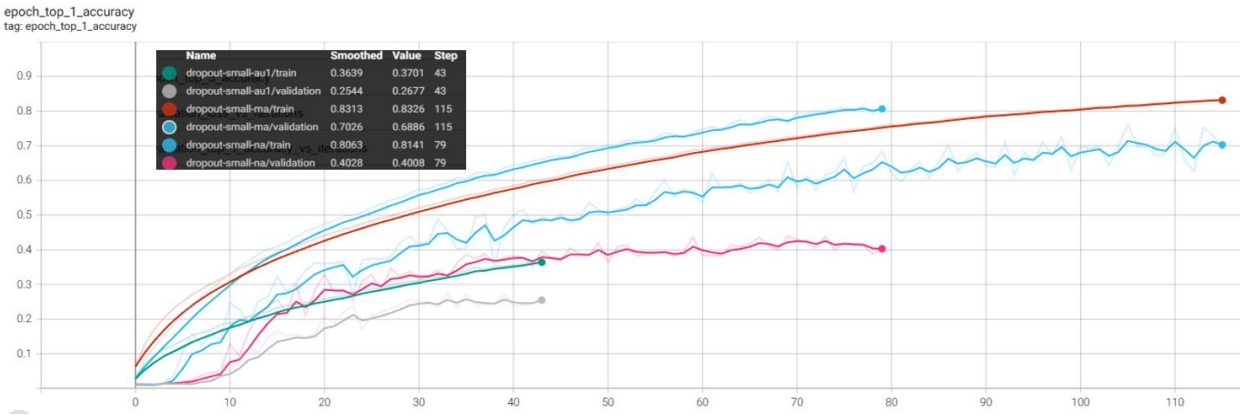


11. dropoutos tanítások hibái

A dropouttal tanított hálóknál nagyobb overfittinget vettem észre mint amire számítottam. Kisebbre számítottam mint a dropout alkalmazása nélküli hálóban , de ez végül nem így történt. Ennek következményeként betanítottam még egy hálót aminél ismét megfeleztem a filterek számát, így megnegyedelve azokat. A háló paramétereinek száma is csökkent már csak 2.8 millió. A kevesebb paraméter célja hogy csökkentse az overfittinget a modell felesleges paramétereinek megszűntetésével. Az így kapott eredmények:



12. Kisebb, dropoutos modell hibái



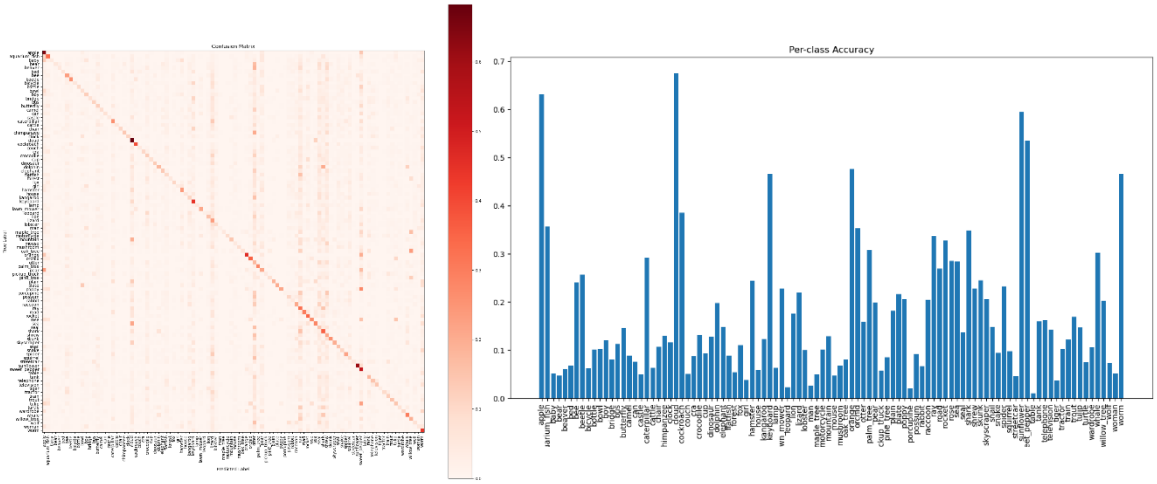
13. Kisebb dropoutos modell Top-1 pontossága

Az így konstruált háló elérte a célját. Csökkentette a túltanulást és még jobb eredményeket ért el, különösen a tanító és validációs halmaz közti különbség csökkentésében. Ami talán még jelentősebb, hogy megközelítőleg ugyanolyan jó eredményeket ért el, mint a nagyobb modellek, a paraméterszám megfelezésével.

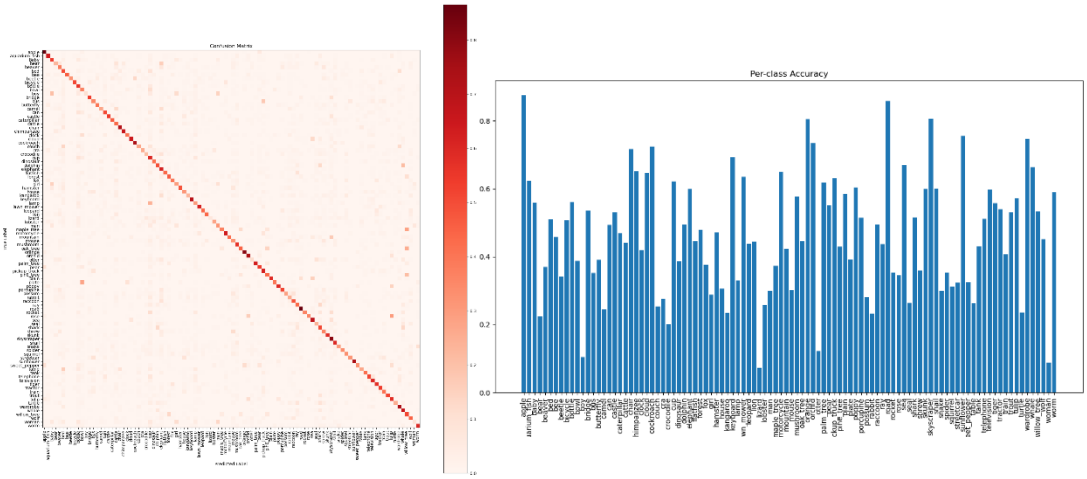
VI. A hálók kiértékelése

Az eddig látott mércék alapján hasonló eredményeket ér el mindegyik háló. De közelebbi vizsgálat alapján új féle különbségeket láthatunk. A modell bizonyossága egy fontos mérce, ami jól kiemeli az adat augmentáció hatásosságát. A hálók kiértékelését nem csak a teszt adatokkal végeztem, hanem a teszt adatokat az augmentációs rétegekkel transzformálva is többször kiértékeltem a hálókat, így láthatóvá válik a háló bizonyossága sok különböző szögből még nem látott adaton. A háló által kapott predikciókat összevetettem a valós eredménnyel, majd az egyes mintákra kapott bizonyosságokat kiátlagoltam, így kapva egy számot, ami jellemzi a háló biztosságát.

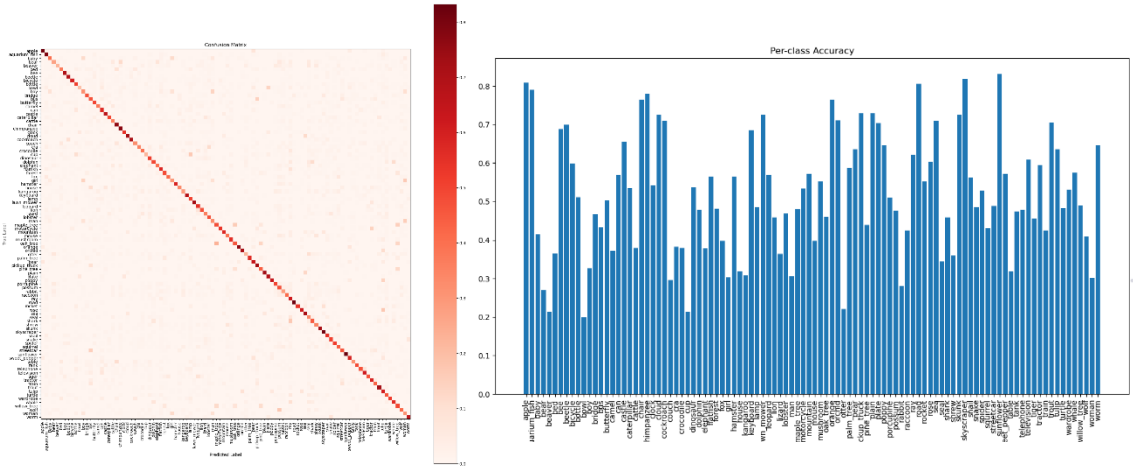
Ezen kívül hasznosnak találtam a tévesztési mátrixot. Itt minden osztálynak láthatjuk minden másik osztállyal való összetévesztését. Ezzel vizsgálhatjuk, hogy a modell mely osztályokat keveri össze egymással, vagy ha van egy osztály, ami kitűnően gyengén van osztályozva. A Per-class Accuracy ábrák az egyes osztályokra kapott pontosságot mutatja, így egyértelműbb képet kapva a modell teljesítményéről. Az eredmény a következő diagramok:



MyInception-NA-hoz tartozó tévesztési mátrix és osztályonkénti pontosság



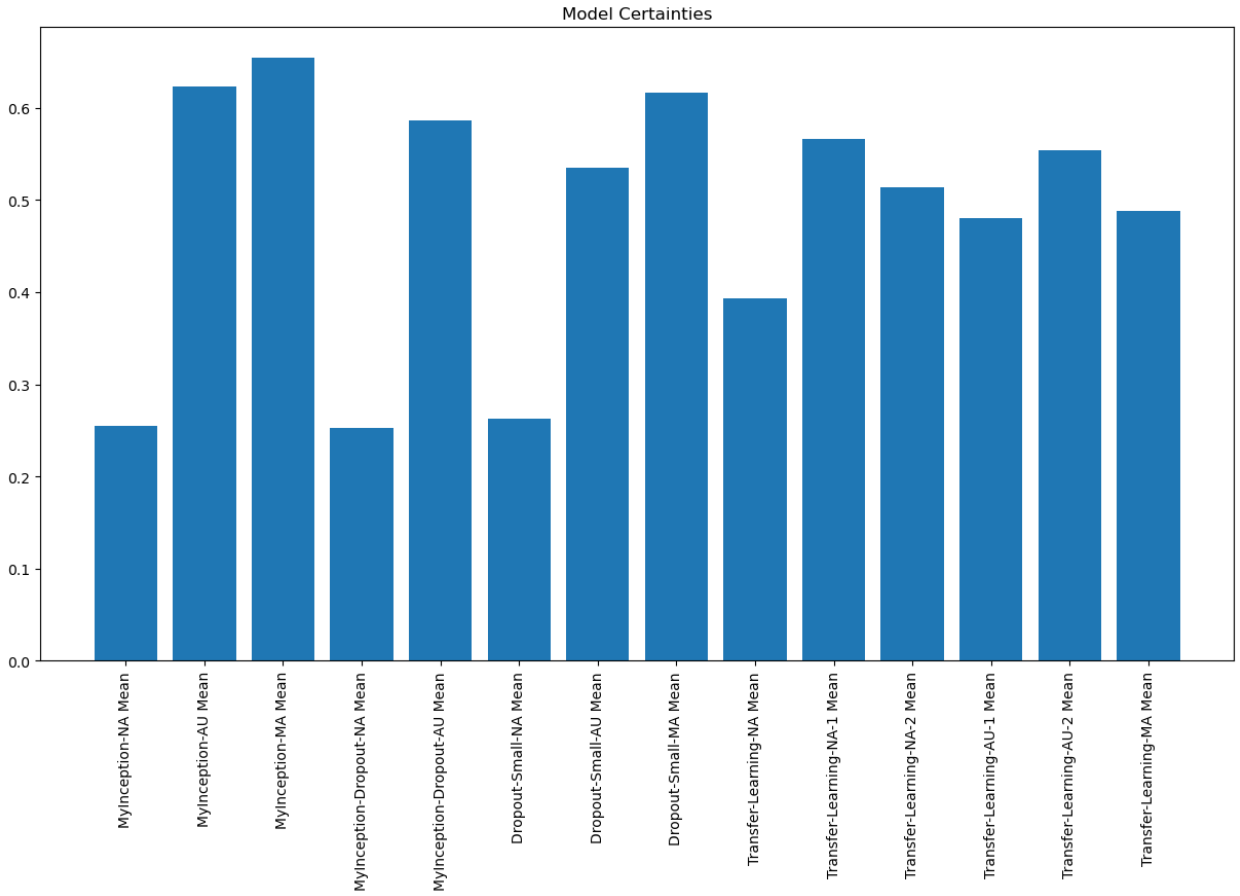
MyInception-AU-hoz tartozó tévesztési mátrix és osztályonkénti pontosság



MyInception-MA-hoz tartozó tévesztési mátrix és osztályonkénti pontosság

Az ábrákon jól látszik az adat augmentálás hatása. Az augmentáció nélküli adatokon tanított modellnél van egy-két kitűnő osztály, de általánosságban nagyon rosszul prediktál, mivel a tanítása során nem látott semmilyen augmentált adatot, így azokon a képeken nagyon rossz a teljesítménye. Amint bevezetjük az augmentációt a háló teljesítménye jelentősen megugrik, a néhány nagy pontosságú osztály helyett azt látjuk, hogy a legtöbb osztályra nagy a pontossága és eltűntek az olyan osztályok, amikre nagyon alacsony a pontosság, azaz szinte egyáltalán nem tudja megmondani a háló, ha azzal találkozik. Ezek az első tévesztési mátrixon a vörösebb oszlopok voltak, de már csak elszórva látunk pontokat, ami két osztály összekeverését mutatja. Amikor még több adattal tanítjuk a modellt, még jobb eredményeket kapunk. Minden osztályra a pontosság körülbelül 10%-kal nőtt és még kevesebb az elmaradó osztály. A tévesztési mátrixon láthatóan „halványabb", ezt azt mutatja, hogy kevesebbet téveszt össze egymással egy-egy osztályt a modell. Így is előfordulnak még olyan osztályok, amiket gyakran összekever egymással a háló, például a tölgy és juharfákat, de a kettő közt a különbséget nem is biztos, hogy egy 32x32-es képről meg lehet tanulni, mivel egy ekkora képen nem biztos, hogy jelen vannak azok a jellemzők, amik alapján ez lehetséges lenne.

A többi modell hasonló javulást mutat, bár más eredményekkel. Az összes diagram megtalálható a projekt GitHub repositoryjában a plots mappa alatt, valamint a betanított modellek a models mappában.



14. Az összes modellre kiszámított bizonyosság

Ezen az ábrán láthatóak a fejezet elején leírt módszerrel kiszámolt bizonyosságok az összes modellre. Ami meglepő, hogy a transfer learninggel tanított modellek nem teljesítenek jobban, mint az általam tanítottak. Ez lehet azért is mert ezekkel a modellekkel kevesebbet tudtam foglalkozni, mert akár 10-15 perces epochokokkal dolgoztam. Lehet, hogyha még több réteget taníthatóvá teszek, akkor jobban tudja finomítani a megtanult reprezentációkat és jobb eredmények ér el. Érdekes még, hogy legjobban az először konstruált modell teljesít, viszont fontos, hogy a Dropout-Small modell alig teljesít rosszabbul, míg fele annyi paramétere van.

VII. Nehézségek a tanítással és további ötletek

A félév során nagy kihívást jelentett robosztus környezetben futtatni a tanításokat. Elsősorban Google Colabot használtam, majd később a Kaggle hasonló megoldássá által nyújtott erőforrásokat is elkezdtem használni. Az egész félév során belefutottam abba, hogy egy tanítás közben megszűnt a környezet inaktivitást miatt, vagy mivel a Colab nem garantált erőforrásokat nyújt, ezért szimplán megszűnt a GPU elérésem és kezdhettem elölről a tanítást. Ugyanezen indok miatt volt, hogy egy hétig egyáltalán nem volt GPU hozzáférésem. Ez kifejezetten frusztráló volt és több tíz órám ment el olyan tanításokra, amik félbe szakadtak és a futtatókörnyezettel együtt a mentett modell is megszűnt létezni. Ez okból hiányzik például a MyInception-Dropout-MA tanítás, mert 3 és fél óra után megszakadt a tanítás és utána sehogy nem tudtam GPU-hoz jutni. Valószínűleg elértem a maximális ingyen felhasználható kvóta tetejét, Kagglen pedig épp egy másik tanítás futott. Tanulságként, megpróbálok legközelebb saját GPU-val egy robusztusabb futtatókörnyezetet kialakítani, bár ez sem egyszerű, mert próbálkoztam vele a félév során, de több fronton meghiúsult.

Végeredményben az általam felépített modellek elég jó eredményeket értek el, de még így is sok javítani valót látok bennük és más technikákat, amiket ki lehet próbálni. Az egyik ilyen amire GPU szűkében sajnos nem volt lehetőségem, az Attention mechanizmus vizsgálata a hálóban. További javításra látok lehetőséget a kisebb dropoutot használó modellben is. A teljesítményt valószínűleg tovább lehetne javítani a stemben végzett durva méret csökkentés mérséklésével és a filterek elosztásának megváltoztatásával.