

BSC SZAKDOLGOZAT

Örvös légykapók énekének gépi tanulás alapú osztályozása

2021. május 31.

Jakobi Ádám

ELTE FIZIKA BSC

Témavezető: dr. Stéger József



Tartalomjegyzék

1. Bevezetés	1
2. Gépi tanulás és mély neurális hálók bevezetés	1
2.1. Gépi tanulás fajtái	3
2.1.1. Felügyelt tanulás (Supervised learning)	3
2.1.2. Nem felügyelt tanulás (Unsupervised learning)	4
2.1.3. Megerősítéssel tanulás (Reinforcement learning)	4
2.2. Teljesen összekapcsolt neurális hálózatok	5
2.3. Aktivációs függvények	6
2.4. A veszteségfüggvény (loss function)	8
2.5. Hibavisszaterjesztés (back propagation)	9
2.6. Adam optimalizáló	11
2.7. Regularizációk	12
2.8. Adatnövelés (data augmentation)	13
2.9. Konvolúciós neurális hálózatok (CNN)	14
2.10. Normalizálás, standardizálás	17
2.11. Mértékek (Metrics)	19
2.12. A tanulás követése	20
2.13. Validációs módszerek	21
3. Madárhang detektáló kihívás	22
3.1. Az adatok	22
3.1.1. Források	22
3.1.2. Az adatok struktúrája	23
3.1.3. A teszhalmaz manuális címkézése	23
3.2. A felvételek feldolgozása	24
3.2.1. Diszkrét Fourier-transzformáció (DFT)	24
3.2.2. Rövid idejű Fourier-transzformáció (STFT)	25
3.2.3. Mel spektrogramok	26
3.2.4. Mel spektrogramok megvalósítása	27
3.3. A hálózat felépítése	29
3.4. A hálózat tanítása	29
3.5. Keretrendszer és futtatás	30
3.6. Validáció és tesztelés	31
3.7. Az eredmények értékelése	31
4. Diszkusszió	32

5. Függelék	33
5.1. Keresztvalidáció 64-es kötegméretre	33
5.2. Keresztvalidáció 32-es kötegméretre	35
5.3. A neurális hálózat felépítése	36

1. Bevezetés

Szakedolgozatom eredeti célja örvös légykapók énekének gépi tanulás alapú osztályozása volt, azonban a munka közben felmerült egy másik projekt is: az ornitológiai kutatócsoportban, ahol dolgoztam arra kértek, hogy reprodukáljam a Bird Audio Detection Challenge¹ győztes megoldásának² neurális hálóját, melynek segítségével képesek detektálni egy hosszú felvétel körülbelül 10 másodpercesre vágott szakaszairól, hogy van-e bennük madárének, még hozzá kereszttanulással. Ez azt jelenti, hogy a neurális hálót különböző helyről mintavételezett felvételekkel tanítjuk be, mint amire később használni akarjuk (például Angliában készült felvételekkel tanítunk és Chernobili madárénekekre tesztelünk). Ezzel jelentősen fel lehetne gyorsítani a felvételek feldolgozását és csökkenteni a hozzá szükséges humán erőforrások mennyiségét. Céлом továbbá, hogy a projekten keresztül beletanuljak a gépi tanulás és mély neuronhálók elméletébe, a megvalósításhoz szükséges eszközök, keretrendszerek, programok használatába és a tudományos életben folytatott további lehetőségek kiaknázása, kutatómunka folytatása, illetve hogy tapasztalatot szerezzek nagyobb, összetett feladatok elvégzésében, és szembesítsem magam a nagy adatsorokra tanítási technikai és időbeli kihívásaival is.

2. Gépi tanulás és mély neurális hálók bevezetés

Az emberiséget már régóta foglalkoztatja, hogyan tudna létrehozni magához hasonló, mesterséges értelmet. Először az élőlények területére korlátozódott ez az érdeklődés, a gépesítés beköszöntével azonban továbbgyűrűzött az emberek által használt eszközökre is. A cél már olyan gépek létrehozása volt, amik képesek elvégezni az emberek helyett a munkát. Ahogy fejlődött a technológia, és feltalálták a számítógépeket, szépen lassan egyre több feladatot tudtak automatizálni, azonban a hagyományos számítógépek, bár rengeteg téren messze képesek voltak túlszárnyalni az embereket, sajnos alkalmatlanok voltak az emberek számára akár egészen egyszerű feladatok elvégzésére, mint a képfelismerés, hangfelismerés, a beszéd, illetve nem voltak képesek az önálló, a helyzettől függő árnyalt gondolkodásra.

Ma minden eljárást, mely lehetővé teszi, hogy számítógépek az emberi viselkedést utánozzák, mesterséges intelligenciának nevezünk. Maga a mesterséges intelligencia kifejezés John McCarthy-tól származik a témának szentelt első konferenciáról (1956). A gépi tanulás a mesterséges intelligencia részhalmaza, itt a számítógép már explicit programozás nélkül is képes tanulni. A gépi tanulás során, a nyers adatokból még az embereknek kellett olyan változókat készíteniük, melyek segítséget nyújtottak az adott probléma megoldásában.

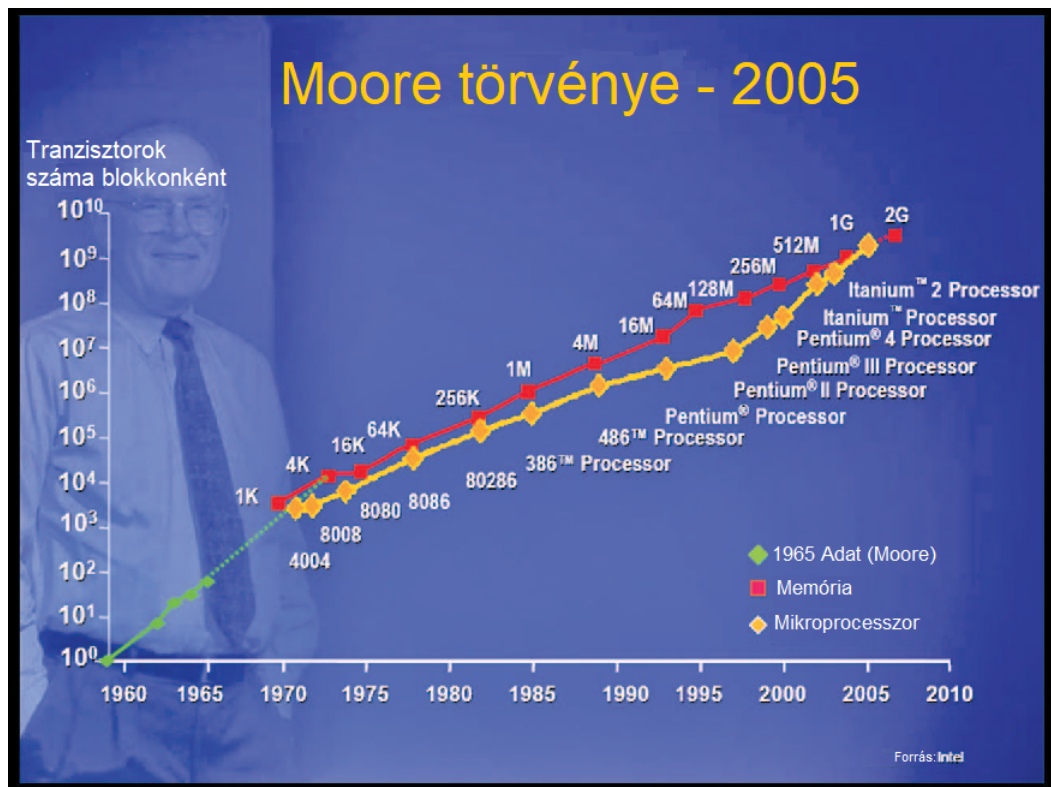
A mély tanulás a gépi tanulás olyan esete, ahol már nem szükséges manuálisan létrehozni a nyers adatsoron kívüli változókat, vagy csak nagyon kis mértékben, mert azokat egy neurális háló segítségével a rendszer maga hozza létre a probléma megoldásához.

Az első, a szakma által mesterséges intelligenciaként elfogadott eredményt Warren McCulloch és Walter Pitts produkálta 1943-ban³, mellyel megalapították a neuronhálózatok elméletét. Munkájuk alapjául a Turing számításelmélet, az ítéletkalkulus és az agyi neuronok működéséről meglévő ismereteik szolgáltak. Egy mesterséges neuronhálózat modelljét javasolták, melyben minden neuronnak két állapota volt lehetséges, vagy bekapcsolt volt, vagy kikapcsolt. Egy neuron akkor került bekapcsolt állapotba, ha kellő számú szomszédos neuron stimulálta. Úgy tartották, hogy egy neuron egyenlő az őt aktiváló logikai állítással, ezenkívül megmutatták, hogy a neurális háló megfelelő konfigurációjával az összes logikai művelet és ismert függvény előállítható.

Donald Hebb egy olyan értékfrissítő szabályt mutatott be a neuronok közötti kapcsolatok erősségének frissítésére, amivel lehetővé válik a háló tanulása⁴. Tanulási szabálya máig érvényes hatású modellnek bizonyult.

Ezt követően, 1951-ben, a Princeton Egyetem matematika tanszékén két végzős hallgató-Marvin Minsky és Dean Edmonds-megépítette az első neurális számítógépet. A hálózat negyven összekapcsolt neuron működését szimulálta.

A gépi tanulás a XX. században azonban elsősorban csak kutatási téma volt. A neuronhálók sikerének az 1990-es években többek között az szabott határt, hogy nem tudták hatékonyan tanítani a több rejtett réteggel rendelkező neuronhálókat. Ezt az időszakot a "neuronhálók telének" is nevezték. Az ezredforduló környékétől kezdve, és különösen a 2010-es években azonban a gépi tanulási megoldások széleskörűen elterjedtek, a képek elemzésétől a gazdasági előrejelzésekig, és mára egy iparág alakult ki körülötte. Ennek legfőbb okai a GPU-k fejlődése, mely egyre gyorsabb futtatást tesz lehetővé a megnövekedett számítási kapacitás következményeként (1. ábra), az összegyűlt nagyméretű adatbázisok (példaul ImageNet), amiket fel lehet használni a hálók tanításához és az új szoftverek, keretrendszerek (Pytorch, Theano, Tensorflow), melyek segítségével sokkal gyorsabban, kevesebb munkával hatékony gépi tanuló algoritmusokat lehet létrehozni.



1. ábra. Az intel processzorok tranzisztorszámainak növekedése 1960-tól az 2005-ig⁵. Megfigyelhető, hogy a növekedés jól illeszkedik Moore törvényének elméleti görbéjére. Ez a technikai fejlődés eredményezte a gépi tanulás, és főleg a mély neurális hálók elterjedését.

Ma már bárki számára elérhetőek a fent említett szoftverkönyvtárak, amelyekkel a gyakorlatban is megvalósítható a gépi tanulás.

2.1. Gépi tanulás fajtái

A gépi tanulás három kategóriába sorolható, a felügyelt, a nem felügyelt és a megerősítő tanulásba.

2.1.1. Felügyelt tanulás (Supervised learning)

A felügyelt tanulás esetében egyszerre állnak rendelkezésre a háló bemeneti paraméterei, illetve a kimenetek, amiket szeretnénk, hogy a háló a bemeneti paraméterekre visszaadjon. A rendezett adatpárok egy kis részét a folyamat elején elkülönítjük későbbi tesztelés céljából, ezt hívjuk teszhalmaznak, ami semmilyen formában nem vehet részt a háló tanításában. A maradékból, többféle felbontási módszerrel előállíthatunk tanító és validációs halmazokat. A tanító halmazt használjuk a háló betanítására, mely során vizsgáljuk a háló bemenetre adott kimenetét, egy veszteségfüggvény segítségével mérjük a várt eredménytől történő eltérést, és egy optimalizáló algoritmust használva frissítjük a háló kapcsolatainak erősségét (az

úgynevezett súlyokat), hogy a későbbi futtatásoknál jobb eredményeket kapjunk. A validációs halmaz adatai nem vesznek részt közvetlenül a tanulásban, hanem a háló pontosságát teszteljük vele abban a fázisban, amikor még nem választottuk meg véglegesen a rendszerünk hiperparamétereit. A hiperparaméterek manuálisan megválasztható, nem tanítható változók, amiket kísérletezések segítségével állítunk be optimálisra. Döntésünket a validációs eredmények befolyásolják, ezért van szükség egy külön tesztalmazra, amin minden befolyástól függetlenül tesztelhetjük az eredményeinket.

A felügyelt tanulást két problémátípus megoldására használjuk. Az egyik az osztályozás, ahol a cél a bemeneti paraméterek alapján egy diszkrét, véges halmazból kiválasztani a megfelelő, a bemenethez tartozó osztályt. A háló kimenete egy vektor, ami nulla és egy közötti valós számokat tartalmaz. A vektor elemei reprezentálják az egyes osztályokat, minél nagyobb egy elem számértéke, annál biztosabban állítja a rendszer, hogy a bemenet abba az osztályba tartozik. Ilyen feladatra példa állatfajták felismerése képek alapján, lehetséges kategóriák például a kutya, macska, kecske.

A másik típus a regresszió, ahol a rendszer egy vagy több folytonos kimeneti értéket szeretne meghatározni a bemeneti paraméterek alapján, például a hőmérsékletet és a páratartalmat.

2.1.2. Nem felügyelt tanulás (Unsupervised learning)

Nem felügyelt tanulás alkalmazásánál nem állnak rendelkezésre osztályok, feliratok, vagy adatok arról, hogy mit várunk a rendszer kimeneténél, pusztán a nyers adatok (és az esetlegesen a nyers adatokból manuálisan létrehozott jellemzők). Azt szeretnénk elérni, hogy az algoritmusunk maga találja meg az adatokban rejlő még ismeretlen mintákat. Nem felügyelt tanulásra példa a klaszterezés, ahol maga a rendszer találja ki, hány és milyen osztályba sorolhatók az adatok és a főkomponens analízis, mely egy olyan koordinátarendszert hoz létre, ahol a tengelyek egymásra merőlegesek és sorban olyan irányúak, hogy a lehető legjobban kifejezzék az adatok varianciáját.

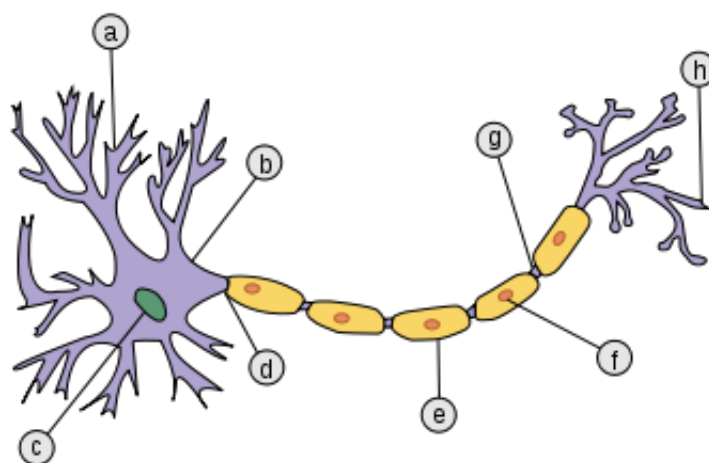
2.1.3. Megerősítéssel tanulás (Reinforcement learning)

A megerősítéssel tanulás az újszülöttek fejlődéséhez hasonlatos. A csecsemő (rendszerünk "ügynöke"/ágense) képes érzékelni a környezetét, cselekedni, döntéseket hozni, azonban nincs konkrétan előírt szabály arról, hogy milyen helyzetben milyen viselkedés lenne elvárt. A viselkedés formálására megerősítéssel alkalmaznak, a kívánt viselkedést jutalmazzák, a nem kívánt viselkedést pedig büntetik, így tanulja meg a rendszerünk, milyen helyzetben milyen cselekvést kell végrehajtania.

Jó példa erre egy megerősítéses tanulás alapú sakk program, ahol a bábuk, vagy akár a játék megnyerése vagy elvesztése az eredménynek megfelelő pontok elnyeréséhez vagy elvesztéséhez vezet.

2.2. Teljesen összekapcsolt neurális hálózatok

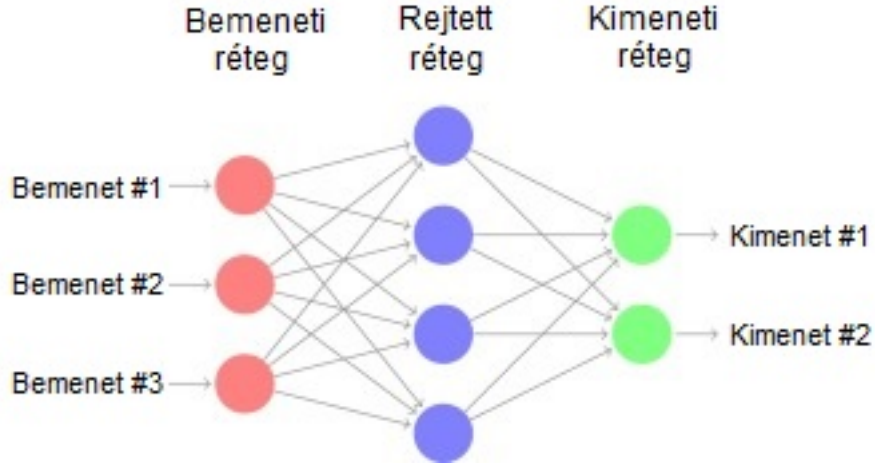
A neurális hálókat az élőlények agyában található idegrendszeri kapcsolatokról mintázták. Az agyban az idegsejtek sejtmagját a sejttest veszi körül, amiből rövid nyúlványok nyúlnak ki, melyeket dendriteknek hívnak. A sejttestből egy hosszú nyúlvány is fakad, az axon, aminek a végén szintén nyúlványok erednek. Ezt hívják végfácskának (telodendrion). A neuronok képesek egymásba kapcsolódni, az axonok végén lévő szálakkal csatlakoznak a dendritek nyúlványaiba, majd pedig külső inger hatására elektromos jeleket továbbítanak egymáson. A folyamat úgy zajlik, hogy egy neuron dendritjein keresztül beérkező elektromos jelek összeadódnak és eljutnak az axondombhoz, ahol ha az összeg meghaladja a szükséges küszöbértéket, akkor a neuron kisül, és elektromos jelet továbbít azoknak a neuronoknak, amikhez ő maga csatlakozik.



2. ábra. Egy neuron vázlatos képe⁶: a. dendrit, b. sejttest, c. sejtmag, d. axondomb, e. Schwann-hüvely, f. Schwann-sejt (mag), g. Ranvier-féle befűződés, h. végfácska (telodendrion) elágazódás

Ehhez hasonlóan próbálták implementálni a mesterséges neurális hálókat is. Az úgynevezett teljesen összekötött (fully connected), avagy sűrű (dense) neurális háló (mely a mély tanulásban használt neuronháló alapját képezi) csúcsokból, másképp neuronokból épül fel. A neuronok rétegekbe (layer) rendeződnek, és minden réteg összes neuronja kapcsolatban áll a vele szomszédos rétegek összes neuronjával. A neurális háló első rétege a bemeneti réteg (input layer), az ezt követő, középső rétegeket hívjuk rejtett rétegeknek (hidden layers), végül az utolsó a kimeneti réteg (output layer). A teljesen összekapcsolt neurális háló rétegeiről a későbbiekben fel-

teszem, hogy vektorok, így az egyes rétegek közötti kapcsolatok erősségei (a súlyok) felfoghatók mátrixok formájában (súlymátrixok).



3. ábra. Sűrű neurális háló lehetséges ábrája⁷, bal oldalt a bemeneti réteg, mely tartalmazza a háló bemeneti vektorát, középen a rejtett rétegek, végül pedig a kimeneti réteg. Ha a háló több mint egy rejtett réteggel rendelkezik, mély neurális hálónak hívjuk.

A mély tanulás tulajdonképpen egy sokszorosán összetett, nemlineáris, sokváltozós függvényillesztési probléma. Minden neuronréteg az őt követő réteg bemenete. Egy rétegben egy neuron értékét úgy kapjuk meg, hogy vesszük a bemeneti réteg lineáris kombinációját, hozzáadunk egy konstans értéket (előfeszültség, angolul bias, tanulható paraméter) és a kapott eredményre egy aktivációs függvényt alkalmazunk, ami egy nemlineáris leképezést valósít meg (az aktivációs függvény teszi a hálózatot nemlineárisra, ezért tudunk neurális hálózatokkal, mint függvényekkel nemlineáris problémákra illeszteni). A fenti példában (3. ábra) a kimeneti vektor kiszámítása a következőképpen nézne ki:

$$\vec{v}_{ki} = \Phi^{(2)}(\widehat{W}^{(2)}\Phi^{(1)}(\widehat{W}^{(1)}\vec{v}_{be} + \vec{b}^{(1)}) + \vec{b}^{(2)}),$$

ahol $\widehat{W}^{(1)}$ a bemeneti és a rejtett réteg, $\widehat{W}^{(2)}$ pedig a rejtett és a kimeneti réteg között elhelyezkedő kapcsolatok súlymátrixa, $\vec{b}^{(1)}$ és $\vec{b}^{(2)}$ az előfeszültségek a rejtett és a kimeneti rétegben, végül pedig $\Phi^{(1)}$ és $\Phi^{(2)}$ az aktivációs függvények (az aktivációs függvényt a beadott vektor minden elemére, elemenként kell alkalmazni, így a visszatérési értéke a bemenettel megegyező hosszú vektor).

2.3. Aktivációs függvények

Az aktivációs függvényeknek sok fajtája ismert, általában az a céljuk, hogy nemlineárisra alakítsák a neurális hálózatot. A legismertebbek közül szeretnék most felsorolni pár példát.

Sigmoid vagy logisztikus függvény

A sigmoid függvény nulla és egy közé képez, képlete:

$$\Phi(x) = \frac{1}{1 + e^{-x}}.$$

Tipikusan a kimeneti réteg aktivációs függvényeként használják klasszifikációs feladatoknál. A rejtett rétegeknél nem tanácsos alkalmazni, mivel nulla és egy közeli értékeknél a függvény deriváltja nagyon kicsi, így a tanulási folyamat jelentősen lelassul (eltűnő gradiens probléma).

Hiperbolikus tangens

A hiperbolikus tangens függvényt a rejtett rétegekben szokták használni, minusz egy és egy közé képez, képlete:

$$\Phi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

Zérus átlagú, így könnyebb vele az optimalizáció, azonban a sigmoidhoz hasonlóan erre a függvényre is jellemzőek az eltűnő gradiensek a függvény két végén.

ReLU (rektifikált lineáris egység)

A ReLU szintén a rejtett rétegekben használható függvény, egyszerű, gyorsan számolható, sokkal gyorsabban konvergál, mint a sigmoid vagy a hiperbolikus tangens függvény. Képlete:

$$\Phi(x) = \max(0, x),$$
$$\frac{d\Phi(x)}{dx} = \begin{cases} 1, & \text{ha } x > 0 \\ 0, & \text{egyébként} \end{cases}.$$

A ReLU hátránya, hogy nincs felső korlátja, így az aktiváció elszállhat. Elkerüli ugyan az eltűnő gradiens problémát, azonban előfordulhat, hogy olyan súly és előfeszültség kombináció alakul ki egy neuronra, hogy a neuron aktivációs függvény előtti kimenete mindig negatív lesz (például nagy, negatív előfeszültség esetén). A ReLU deriváltja $x < 0$ esetén zérus, így az adott neuronhoz tartozó súlyok nem fognak többé megváltozni, ezért a neuron sem fog a későbbiekben aktiválódni, úgynevezett "halott" neuron lesz.

Szivárgó (leaky) ReLU

A szivárgó ReLU hasonlóan jó, mint a ReLU, számítása csak egy kicsivel tart tovább, cserébe viszont megoldja a "halott" neuronok problémáját azzal, hogy egy

tetszőlegesen kicsi, α meredekségű egyenest használ $x < 0$ esetén, így a függvény gradiense negatív értékek esetén sem fog teljesen eltűnni. A szivárgó ReLU aktivációs függvény képletekkel kifejezve:

$$\Phi(x) = \max(\alpha x, x),$$

$$\frac{d\Phi(x)}{dx} = \begin{cases} 1, & \text{ha } x > 0 \\ \alpha x, & \text{egyébként} \end{cases}.$$

Softmax

A softmax függvényt többkategóriás osztályozási problémák esetén használják, minden osztályra valószínűségi értékkel tér vissza:

$$\Phi(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

2.4. A veszteségfüggvény (loss function)

A veszteségfüggvény egy deriválható függvény, amit arra használunk, hogy objektív számértéket kapjunk a hálózat hibájáról a kimenet és a várt kimenet (ground truth érték) ismeretében. Minél nagyobb az eltérés, annál nagyobb lesz a veszteségfüggvény értéke. A tanítás célja, hogy minimalizáljuk a veszteségfüggvényt. A leggyakrabban alkalmazott veszteségfüggvények a következők ($(x^{(i)}, y^{(i)})$ az összetartozó adat-felirat párokat jelöli, h_W a W súlyokat tartalmazó leképezés, $J(W)$ a veszteségfüggvény, M pedig az adatpárok elemszáma):

Átlagos négyzetes eltérés (mean squared error, MSE)

$$J(W) = \frac{1}{2M} \sum_{i=1}^M (h_W(x^{(i)}) - y^{(i)})^2$$

$$\frac{\partial J(W)}{\partial W_j} = \frac{1}{M} \sum_{i=1}^M (h_W(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Bináris kereszt-entrópia (binary cross-entropy)

$$J(W) = \frac{1}{M} \sum_{i=1}^M [-y^{(i)} \log(h_W(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_W(x^{(i)}))],$$

$$\frac{\partial J(W)}{\partial W_j} = \frac{1}{M} \sum_{i=1}^M (h_W(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Többkategóriás kereszt-entrópia (multiclass cross-entropy)

$$J(W) = - \sum_{i=1}^M \sum_{j=1}^N y_j^{(i)} \log(h_W(x^{(i)}))_j,$$

ahol N a lehetséges kategóriák száma.

2.5. Hibavisszaterjesztés (back propagation)

A háló tanításához szükség van a súlyok módosítására, melyeket kezdetben véletlenszerű értékekkel inicializálunk. A módosítást úgy kell végeznünk, hogy utána pontosabban működjön a rendszerünk. Ahogy az előző részben említettem, erre használható a veszteségfüggvény. Ki kell számolni a veszteségfüggvény gradiensét a tanulható paraméterek függvényében, és a tanulható változókból le kell vonni az adott változóhoz tartozó gradiens komponensét, beszorozva a tanulási ráta (learning rate) értékével (a tanulási ráta rendszerünk egy hiperparamétere, mely a hálósúlyok frissítésének mértékét határozza meg).

A gradiens kiszámításához parciális deriválásokra és a lánc deriválási szabály alkalmazására van szükség(ahogy a 4. ábrán látható):

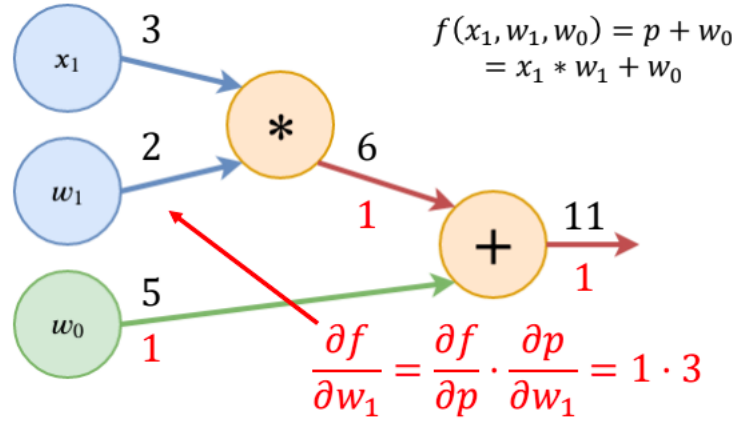
$$\frac{\partial L(\theta)}{\partial \theta_1} = \frac{\partial L(\theta)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial \theta_1} = \frac{\partial L(\theta)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial q} * \frac{\partial q}{\partial \theta_1}$$

$$\frac{\partial L(\theta)}{\partial \theta_2} = \frac{\partial L(\theta)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial \theta_2}$$

Láncszabály

4. ábra. Lánc szabály alkalmazása.⁸ A parciális deriváltakat a függvény kimenetétől kezdve számoljuk vissza.

Eredetileg a gradienst elemenként külön-külön számolták, ami rengeteg időbe telt. Ehelyett a gradiens kiszámításának jelenleg ismert leghatékonyabb módja a hibavisszaterjesztés (angolul backpropagation) módszere. A parciális deriválásokat a veszteségfüggvényre, a háló kimenetétől kezdve visszafele, rétegről rétegre, műveletről műveletre haladva végezzük el úgy, hogy a hátulról kiszámolt gradiens komponenseket eltároljuk, így minden szükséges számítást csak egyszer kell elvégeznünk. Egy egyszerű példa tekinthető meg a 5. ábrán.



5. ábra. Egyszerű példa a hibavisszaterjesztés folyamatára.⁸

A háló frissítésére ezután három lehetőségünk van:

Gradiens ereszkedés (gradient descent)

A teljes adathalmazra kiszámoljuk az összes gradienst, és azok átlagát használjuk:

$$W_t = W_{t-1} - \alpha \nabla L(W),$$

ahol W a tanítható paraméterek listája, α a tanulási ráta és $\nabla L(W)$ a veszteségfüggvény gradienseinek átlaga:

$$\nabla L(W) = \nabla \frac{1}{N} \sum_{n=1}^N l(f(W, x^{(n)}), y^{(n)}).$$

A gradiens ereszkedés kis lépésszám mellett konvergál, de mivel nincsenek benne véletlenszerű ugrások (mindig az összes adatra nézve legnagyobb gradienssel ellentétesen mozdul) könnyen beleragadhat egy lokális minimumba. Ezen kívül nagy memóriaigénye van (egyszerre kell feldolgozni az összes adatot) és sokáig tart, amíg a hálón frissítés történik (ismét azért, mert az összes adatot fel kell dolgozni egy frissítéshez, ami általában már nem fér be a GPU memóriájába, amivel a műveletek egyszerre elvégezhetővé válnának).

Sztochasztikus gradiens ereszkedés

Annyiban tér el a gradiens ereszkedéstől, hogy itt minden adatpár kiértékelése után frissítjük a hálónk súlyait:

$$W_t = W_{t-1} - \alpha \nabla l(W),$$

ahol W a tanítható paraméterek listája, α a tanulási ráta és $\nabla l(W)$ a veszteségfügg-

vény gradiense egy adatpárra. A módszer előnye, hogy kis memóriaigényű folyamat, a frissítések is időben gyakoriak, azonban a frissítés irányai nagyon zajosak, hiszen egy adatpár nem képes reprezentálni az egész adathalmaz tulajdonságait, így a konvergencia lelassul. Azt azonban érdemes megjegyezni, hogy a zajos frissítéseknek generalizáló hatása van, vagyis segít elérni, hogy a háló minél változatosabb új adatokra is megfelelő kimenetet adjon.

Kötegelt gradiens ereszkedés (mini-batch gradient descent)

Ez a frissítési módszer az előző kettő kompromisszuma, a memóriaigénye és a frissítések száma időben az előző két eljárás között van. A frissítések nem olyan kaotikusak, mint a sztochasztikus esetben, azonban még mindig van bennük annyi zaj, ami akár elő is segítheti, hogy a súlyok bekonvergálhassanak a globális minimumba. A kötegméret (batch size) egy hiperparaméter, amit a problémánk függvényében érdemes megválasztanunk. Tipikusan kettőhatvány, általában 32-nek vagy 64-nek választják. Ezen kívül a generalizálás érdekében minden tanulási ciklus előtt megszokták keverni az adatok sorrendjét, így minden körben más elemek kerülnek egy kötegbe és más sorrendben dolgozza fel őket a háló.

Általában ezt a módszert használják a modellek tanítására. A hálót egy köteg feldolgozása után frissítik, egy ilyen lépést iterációnak hívnak. Egy teljes tanulási ciklust, ami során a modell az összes adatot felhasználta a tanulásra, epochnak nevezünk. Egy epochban a lépések száma a tanulási adathalmaz nagyságának és a köteg méretének felfele kerekített hányadosa. Például ha 137 tanító adatom van, és kötegenként 42 elemet használok a tanításra, akkor egy epoch négy lépésből fog állni. Az epochok száma szintén hiperparaméter, melyet manuálisan kell beállítani.

2.6. Adam optimalizáló

Az optimalizálók célja, hogy a veszteségfüggvény a súlyok függvényében minél gyorsabban minimumba konvergáljon. Az optimalizálók közül az úgynevezett Adam optimalizálót szeretném kiemelni és részletezni, ami jelenleg az egyik legjobb és leggyakrabban alkalmazott módszer. Az Adam algoritmus két másik optimalizáló, az AdaGrad és az RMSProp legjobb tulajdonságait felhasználva egészíti ki a fent említett gradiens ereszkedés módszereket:

- Eltárolja a gradiensek mozgó átlagát (momentumát), melynek használatával felgyorsítja a keresést a minimum irányában (AdaGrad),
- Eltárolja a gradiensek négyzetének mozgó átlagát, mellyel a tanulási rátát befolyásolva lelassítja a keresést az oszcilláció irányában (RMSProp).

Képletekben kifejezve az Adam optimalizáló a következőképpen néz ki:

$$v_{t+1} = \beta_1 \cdot v_t + (1 - \beta_1) \nabla_W L(W_t),$$

$$r_{t+1} = \beta_2 \cdot r_t + (1 - \beta_2) (\nabla_W L(W_t))^2,$$

$$\hat{v}_{t+1} = \frac{v_{t+1}}{1 - \beta_1^t},$$

$$\hat{r}_{t+1} = \frac{r_{t+1}}{1 - \beta_2^t},$$

$$W_{t+1} = W_t - \frac{\alpha}{\sqrt{\hat{r}_{t+1}} + \epsilon} \hat{v}_{t+1},$$

ahol v a gradiens mozgó átlaga, r a gradiens négyzetének mozgó átlaga, β_1 és β_2 korrekciós együtthatók, W a neurális háló tanítható paraméterei (súlyai), α a tanulási ráta, ϵ pedig egy kicsi konstans, a nullával osztás elkerülése érdekében. α , β_1 és β_2 hiperparaméterek, tipikus értékeik $\alpha \in [10^{-3}, 10^{-5}]$, $\beta_1 = 0.9$, $\beta_2 = 0.999$.

2.7. Regularizációk

A regularizációs technikák⁹ használatával igyekszünk elkerülni a túltanulás jelenségét, amikor a modell az általános tulajdonságok mellett a tanító halmaz adatainak egyéni jellemzőire is rátanul. Túltanulást eredményezhet egy, a kelleténél nagyobb modell (kicsi modellt használva viszont nem biztos, hogy reprezentálni tudjuk az adatsorunk tulajdonságait), vagy a nem elegendő adat. Eredménye, hogy a modellünk a tanító halmazon ugyan jól teljesít, de a teszhalmazon, vagy új, ismeretlen adatokon nem lesz jó a teljesítménye. A következő alfejezetekben felsorolok pár módszert, amit a túltanulás elkerülésére alkalmaznak.

L1

Az $L1$ regularizáció a veszteségfüggvény (loss function) kiegészítése a súlyvektor 1-es normájával, "Lasso" regularizációnak is hívják:

$$L(W) = l(W) + \lambda \cdot \sum |w_i|.$$

Az $L1$ erősen nulla közelébe kényszeríti a súlyokat, akár teljesen inaktiválhat neuronokat. A neuronok kimenetére alkalmazva teljesen összekapcsolt hálózatból ritka ("sparse") hálóstruktúrát hoz létre.

L2

Az $L2$ regularizáció a veszteségfüggvény (loss function) kiegészítése a súlyvektor 2-es normájával (angolul "weight decay"-nek is hívják):

$$L(W) = l(W) + \frac{\lambda}{2} \cdot \sum w_i^2.$$

Az $L2$ nem regularizál olyan agresszívan, mint az $L1$, a súlyok kis értékeire törekszik.

Kihagyás (dropout)

Minden bemenetnél véletlenszerűen kiválasztjuk a neuronok egy előre megadott százalékát és kinullázzuk őket. Új bemenetnél újraválasztjuk a zérusnak szánt neuronokat. Ezzel arra sarkalljuk a modellt, hogy a neuronok önállóan is tanuljanak és kevésbé hagyatkozzanak a szomszédaikra. Megmutatható, hogy olyan hatása van, mintha a háló összes részhálóját tanítanánk egyszerre és összegeznénk őket. Hátránya, hogy lassítja a tanulást, akár 5-10-szer többször is végig kell menni az adatokon. Ezenkívül egy tetszőleges neuronnak a mögötte lévő neuronréteg kieső elemei miatt kevesebb bemenete lesz, így a lineáris kombináció eredményét be kell még szorozni a megmaradt neuronok arányának reciprokával, hogy az összeg hasonló skálára essen.

2.8. Adatnövelés (data augmentation)

Az adat augmentáció nem tartozik a regularizációs módszerek családjába, de hasonló eredményt érhetünk el a használatával. A tanítás minden lépésében elvégezhetünk egy véletlenszerű transzformációt a tanítás bemenetein, ezzel elérve, hogy a háló a tanítóhalmaz elemeit más környezetben, kisebb változtatásoknak alávetve is képes legyen felismerni. Így a modellünk jobban generalizál, jobban teljesít új, ismeretlen adatokon. Ilyen transzformációk például a képek eltolása, forgatása, kicsinyítése, nagyítása, vagy hangfelvételeknél a felvétel gyorsítása, lassítása, vagy fehér zaj hozzáadása. Ezzel valójában nem növeltük meg az adathalmazunk méretét, csak minden tanítási ciklus előtt elvégeztünk a meglévő adatainkon egy transzformációt és az éppen aktuális transzformált adatokat használtuk a modell tanítására.

2.9. Konvolúciós neurális hálózatok (CNN)

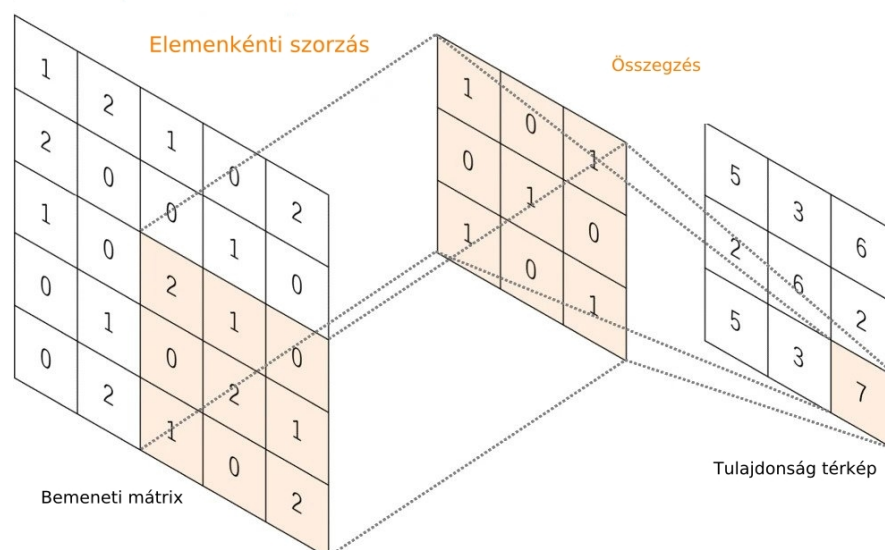
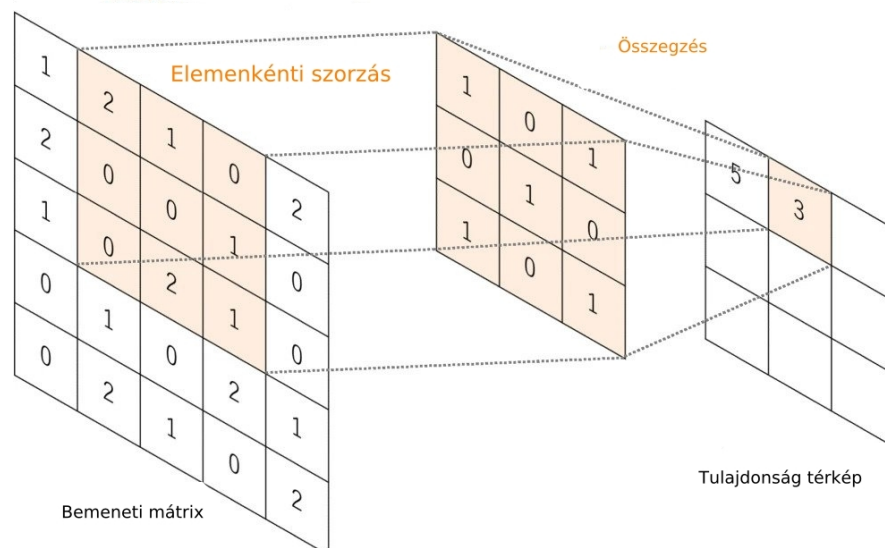
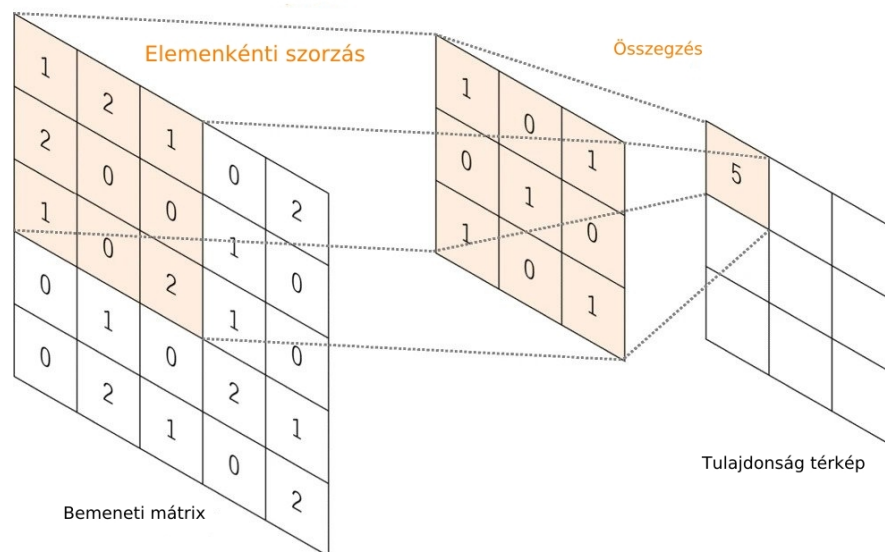
A teljesen összekapcsolt hálózatok nem veszik figyelembe az adatok pozícióját. Képek felismerésére ugyanolyan jól rá tudnak tanulni akkor is, ha a képek pixeleinek pozícióit szisztematikusan összekeverjük, vagyis minden bemenetet azonos módon transzformálunk. Képek esetén a pixelek elrendezésének fontos szerepe van az agyunk számára. Ezen kívül elérhetőek kutatások arra vonatkozólag, hogy az emlősök agya hierarchikusan dolgozza fel a látott információkat. Macskákon végzett kísérletek során megfigyelték, hogy különböző neuronok tüzelnek a vizuális agykéregben egyszerű vízszintes, vagy függőleges élek látványára.¹⁰

A konvolúciós neurális hálózatok konvolúciós rétegei a bemeneti adataikat lokálisan végigpásztázva dolgozzák fel, például képek esetén egyszerre csak kis részletekre koncentrálnak. A kinyert információkat hierarchikusan továbbítják a következő rétegnek. A magasabb rétegek egyre nagyobb lokális részeket fednek le, de felbontásuk eközben folyamatosan romlik, a részletek egyre kevésbé lesznek fontosak. Az objektumok pontos pozíciója kevésbé fog számítani.

A konvolúciós rétegek a képfeldolgozásban használt konvolúcióhoz hasonlóan működnek. Egy darab színcsatorna esetén a bemeneti kép felfogható egy kétdimenziós mátrixként. Ebben az esetben a konvolúciós réteg neuronjainak értékét úgy kapjuk, hogy a réteg tanulható súlyait egy keretben helyezük el, amit kernelnek vagy szűrőnek (filter) is neveznek. A kernelt végigfuttatjuk az összes, a bemeneti kép kernellel azonos dimenziójú részletén. Minden átfedés alkalmával elvégezzük a kernel és a kép részletének Hadamard-szorzatát (más néven Schur-szorzat). A Hadamard-szorzat elemenkénti mátrixszorzást jelent, képlete:

$$C_{ij} = A_{ij}B_{ij}.$$

A kapott tenzor elemeinek összege lesz az adott képrészlethez tartozó neuron értéke, melyre még egy aktivációs függvényt is alkalmazunk. Minden neuron kiszámításához azonos súlymátrixot használunk, ezt súly megosztásnak (weight sharing) hívják. A konvolúció folyamatát szemlélteti a 6. ábra.



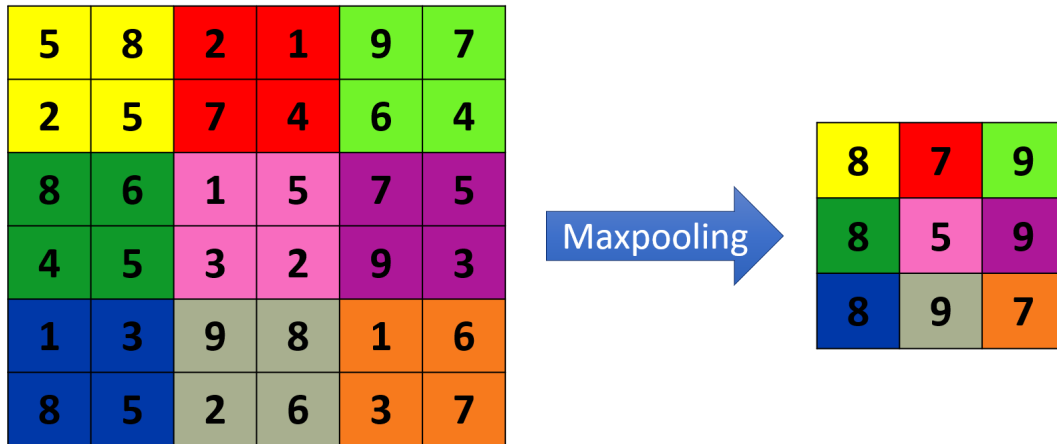
6. ábra. A konvolúció képfeldolgozási művelet működésének szemléltetése.¹¹

Egy egynél nagyobb elemszámú kernel esetén nem lehetséges annyi különböző területet lefedni, mint az eredeti kép pixeleinek száma, így egy hagyományos konvolúciós réteg esetén a neuronok száma kevesebb lesz, mint a bemeneti kép pixelszáma. Bizonyos esetekben ezt szeretnénk elkerülni, ezért a kép szélein valamilyen bővítési eljárást (padding) alkalmazunk, hogy a kép dimenziója ne változzon meg a konvolúció hatására. Erre az egyik gyakran alkalmazott megoldás az, amikor nullákkal bővítik a képek széleit (zero padding). Olyan is előfordul, amikor a szűrő által bejárt tartományt akarjuk korlátozni. Ilyenkor definiálhatjuk a kernel lépésközét (stride). Ha a lépésköz mondjuk kettő, akkor az azt jelenti, hogy csak minden második lehetséges pozícióra fogjuk alkalmazni a kernelünket.

A konvolúciós rétegek célja, hogy képesek legyenek a képekből kivonni azok lényeges tulajdonságait. Minden kernel egy feladatot tud ellátni egyszerre. Ennélfogva egy konvolúciós rétegben párhuzamosan több kernelt használunk, mindegyik más tulajdonság kifejezésére tanul rá. Így egy konvolúciós réteg kimenete általában egy több színcsatornás kép, vagyis egy háromdimenziós tenzor. A csatornák számát hívjuk a kimenet mélységének (depth). Ha a réteg bemenete is több csatornás tenzor, akkor a kernel nem kétdimenziós mátrix, hanem a bemenet mélységével megegyező háromdimenziós tenzor. A Hadamard-szorzást a kép pixelei és a kernel súlyai között az előzőhöz hasonlóan, az azonos indexű elemeken kell elvégezni. A művelethez tartozó neuron értéke szintén egy aktivációs függvény eredménye lesz, melynek bemenete a szorzat tenzor elemeinek összege. Egy réteg kimenetét tulajdonság térképnek (feature map) nevezzük. A konvolúciós réteg lehetséges hiperparaméterei a kernelszélességek (a kernelek általában páratlan oldalhosszúságú négyzetek), a lépésköz és a bővítés mértéke (általában nem alkalmaznak bővítést, tehát a képek szélességének dimenziói konvolúciós régekenként csökkenő).

A háló rétegein felfelé haladva egyre kevésbé akarjuk megőrizni a finom részleteket. Ha az egyik neuron jelzett, hogy észlelt egy objektumot, akkor az objektum pontos pozíciójára később már nem lesz szükségünk. Ezt a műveletet valósítja meg a neuronok összevonása (pooling). Az egyik ilyen módszer a neuronok maximumainak összevonása (max pooling), mely során a képet kisebb egységekre bontjuk, és a következő réteg egy neuronértékének egy ilyen egység maximumát választjuk. Ezt a folyamatot szemlélteti a 7. ábra.

A maximumok összevonása



7. ábra. A maximumok összevonásának egy lehetséges ábrázolása. A maximumok összevonásának hatására továbbcsökken a kimenő mátrix mérete, ami csökkenti a paraméterek számát, gyorsítja a számítási időt és segíti a hierarchikus feldolgozást.

A konvolúciós és összevonó rétegek egymásutáni alkalmazásával ki tudjuk vonni az eredeti adatok hierarchikus tulajdonságait. A hálózat korai rétegei kezdetleges, a későbbiek pedig egyre absztraktabb fogalmakat tanulnak meg. A konvolúciós és összevonó rétegek végén, ahol a paraméterek száma már jelentősen redukálódott, egy teljesen összekapcsolt hálózat segítségével osztályozási vagy regressziós feladatokat tudunk ellátni. A konvolúciós neurális hálók tipikus alkalmazási területe az alakfelismerés.¹²

2.10. Normalizálás, standardizálás

Az adatokat érdemes előfeldolgozni, hogy gyorsabbá tehesük a numerikus műveleteket és könnyebben optimalizálhatóvá tehesük a hálónk tanulását. Az előfeldolgozás egyik fontos célja elérni, hogy a veszteségfüggvény (osztályozási probléma esetén) ne legyen annyira érzékeny a súlyok kis változásaira. Erre az egyik megoldás a normalizáció, ami az adatpontok értékeit a $[0, 1]$ intervallum közé szorítja. A normalizálás érzékeny a kiugró értékekre (outliers), ezért alkalmazása előtt fontos az adatok megfelelő vizsgálata. Egyik lehetséges megvalósítása a normalizálásnak a MinMax skálázás:

$$x'_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}.$$

A másik bevett módszer a standardizálás, melynek során az adatainkat először nulla átlagúvá transzformáljuk, majd pedig egységnyi szórásúvá. A standardizálás a

következő képlettel írható le:

$$Z = \frac{x - \mu}{\sigma},$$

ahol μ az adatok átlaga, σ pedig a szórása. A standardizálás kevésbé érzékeny a kiugró értékekre, jól alkalmazható normalizálás helyett, amikor az adatok letisztítása nehézkes.

Köteg normalizáció (batch normalization)

A hálózat rétegei között, mindig egyszerre csak egy aktuális köteg adatra alkalmazva a standardizálást kapjuk a köteg normalizáció nevű eljárást. A háló rétegeinek aktivációit tudjuk ideális állapotba hozni a használatával. A folyamat a következőképpen fejezhető ki:

$$\begin{aligned}\mu_B &= \frac{1}{m} \sum_{i=1}^m x_i, \\ \sigma_B^2 &= \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2, \\ \hat{x}_i &= \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}, \\ y_i &= \gamma \hat{x}_i + \beta,\end{aligned}$$

ahol m egy köteg elemeinek száma, μ_B és σ_B^2 egy köteg átlaga és szórásnégyzete, γ és β pedig tanítható paraméterek, melyek egy lineáris transzformációt tudnak elvégezni a standardizáló függvényen. Alkalmazása erősíti a hálót, segít generalizálni, regularizáló hatása van, gyorsítja a tanítást és a γ (scale) és β (shift) paraméterek segítenek megőrizni a rendszer stabilitását. A köteg normalizáció máshogy viselkedik tanítás és tesztelés során. Az előbbi esetében az adott köteg adatait használja standardizálásra, tesztelés esetén azonban a köteg átlagok és szórásnégyzetek tanítás alatt eltárolt mozgó átlagait alkalmazza.

2.11. Mértékek (Metrics)

A mértékfüggvény a rendszerünk pontosságának számértékét adja meg. Ez a veszteségfüggvényre is igaz, de a veszteségfüggvény kimenete nem hasonlítható össze univerzálisan más problémák és gépi tanuló algoritmusok veszteségfüggvény értékeivel, a mértékfüggvényekkel ellentétben. A mértékfüggvények általában nem differenciálhatóak, ha mégis, akkor alkalmazhatók veszteségfüggvényként is (megfelelő előjellel véve). A mértékfüggvényeket a tanulás ellenőrzésére és tesztelésre használjuk. Sok ismert változat áll rendelkezésre, a következő alfejezetekben bemutatom ennek pár fajtáját.

Átlagos négyzetes eltérés (mean squared error, MSE)

$$MSE = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2.$$

Az MSE regressziós feladatok pontosságának mérésére használható, illetve egyben veszteségfüggvényként is. Minél kisebb az értéke, annál pontosabb a modellünk.

Klasszifikációs pontosság (classification accuracy)

A klasszifikációs pontosság módszer a helyesen eltalált osztályok százalékát adja meg, kiszámítási módja a helyesen osztályozott elemek száma osztva az összes elemmel, szorozva százszal. A legegyszerűbb mértékfüggvény.

ROC görbe alatti terület (area under the ROC curve, AUC)

A ROC görbe alatti területet bináris klasszifikációs probléma esetén szokták alkalmazni, ami azt jelenti, hogy egy lehetséges osztály jelenlétének vagy hiányának megállapítása a cél. Az igaz pozitív ráta (True Positive Rate, TPR) és a hamis pozitív ráta (False Positive Rate, FPR) tengelyek által határolt görbe alatti terület. A TPR és az FPR képletekkel kifejezve:

$$TPR = \frac{TP}{TP + FN},$$

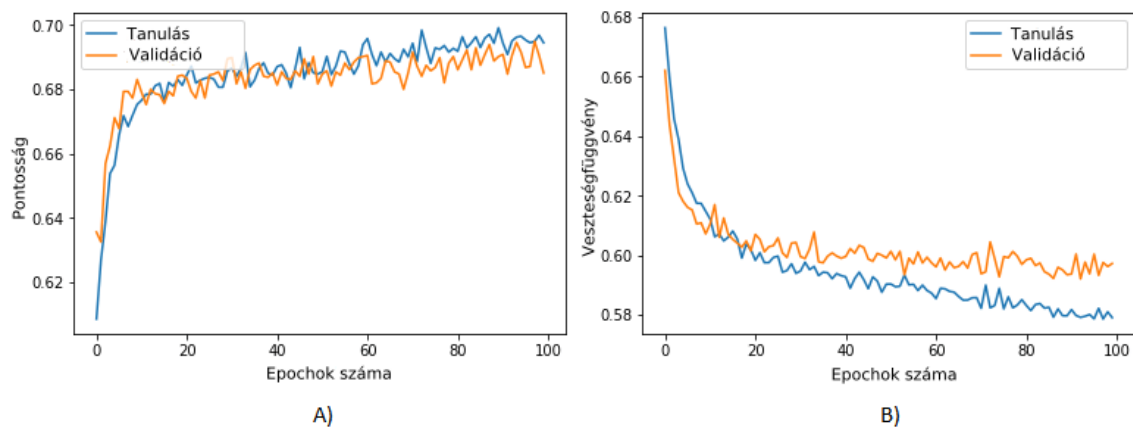
$$FPR = \frac{FP}{TN + FP}.$$

Az AUC kiszámításához sávokra kell osztani a kimenetek lehetséges nulla és egy közötti intervallumát. Minden kiválasztott sáv (például 0.4) alatti értéket negatívnak, fölöttit pedig pozitívnak veszünk. Minden sávot leképezünk az FPR-TPR koordinátarendszerben, és a kapott függvény alatti területet kiszámítva állapítjuk meg a modellünk pontosságát. Ha a pontosság 0.5, akkor teljesen véletlenszerű, hogy a

háló melyik elemet melyik osztályba fogja sorolni. Ha 0, akkor a modellünk a tényleges címkéknek a teljes ellentétét fogja visszaadni, a legjobb, hibátlan eredményt pedig az 1-es érték jelenti.

2.12. A tanulás követése

Ha epochonként kiszámoljuk a tanító és validációs adatok veszteség- és mértékfüggvényeit, akkor lehetőségünk van megfigyelni és ábrázolni a tanulás folyamatát. Erre mutat példát a 8. ábra.



8. ábra. Tanulási görbék lehetséges ábrázolása.¹³

A tanulás megfelelő üteméhez a tanulási ráta tanulás alatti változtatására is szükség lehet. Ehhez rendelkezésre állnak különböző tanulási ráta ütemező módszerek (learning rate schedule), amikkel az epochok haladtával csökkenthető a tanulási ráta. Ilyen megközelítések például:

- Tanulási ráta ejtés (learning rate drop):

Az epochok számához, vagy veszteségfüggvény/mértékfüggvény változásához köthető. A kiváltó ok hatására a program frissíti a tanulási ráta értékét.

- Ütemező függvény használata:

A tanulási ráta értéke változhat az epochok számának függvényében, például lineáris, exponenciális vagy logaritmikus függvényeket használva.

A tanulási görbék figyelésével akár az eredetileg kitűzött epoch szám előtt is leállíthatjuk a folyamatot, ha látható, hogy a modell kezd túltanulni, vagyis hogy a tanító veszteségfüggvény javulása ellenére a validációs pontosság romlásnak indul.

2.13. Validációs módszerek

A validációs módszerek célja a modellünk pontosságának meghatározása. A validációt a tanulási folyamat során használjuk, és a hiperparaméterek finomhangolásához. A végleges modellünk pontosságának mérésére a külön erre a célra, még a tanítás előtt félre tett teszt adathalmazt kell használni. Az egyik lehetséges validációs technika a kihagyás (holdout). Itt a tanító halmaz valamekkora (például 20) százalékát félretesszük és csak validációs célból használjuk, közvetlen tanulásban nem vesz részt. Ebben az esetben viszont fennáll annak a veszélye, hogy a különböző adatosztályok nem egyenletesen oszlanak el a tanító és a validációs halmazokban. Ennek orvoslása érdekében úgy is kettéválaszthatók az adatok, hogy közel egyenlő arányban forduljanak elő a különböző osztályok.

Egy másik módszer a k -szoros kereszt-validáció. Ennél az eljárásnál a tanító halmazt k darab, közel egyenlő részre osztjuk. Ezután k -szor elvégezzük a modell tanítását úgy, hogy minden esetben egy halmazt kiválasztunk validációs halmaznak, a többire pedig elvégezzük a háló tanítását. Nagyon fontos, hogy minden tanításra új modellt kell használni, egy modell nem tanulhat rá egyszerre az egész adathalmazra. A validációnk eredménye a k darab modell validációjának az átlaga. Ezt a módszert akkor érdemes alkalmazni, ha kevés adat áll rendelkezésre (minél kevesebb adatot szeretnénk a validációra fordítani), illetve ha nem okoz időbeli problémát a háló többszöri betanítása. k -t általában 5-nek vagy 10-nek választják. A többszöri betanítás a modelleknél általában egymástól eltérő lokális minimumokat eredményez, így a modellek egymástól eltérő eredményeket adnak. A k -szoros kereszt-validációban betanított modellek felhasználhatók model sokaságként (model ensemble) is. Ennek a módszernek a lényege, hogy egy bemeneti értékre mindegyik modell számol egy kimenetet, melyeknek az átlagát véve pontosabb eredményt kaphatunk, mint a külön-külön eredményekből.

3. Madárhang detektáló kihívás

A verseny célja madarak akusztikus, automatikus detektációja mély tanulás segítségével¹. A madarak jelenlétének vizsgálata fontos szerepet tölt be az ökoszisztéma egészségének megállapításában. A legtöbb madár felismerhető a hangja alapján, így a passzívan végzett hangfelvételek készítése nagymértékben megfelel a madarak állományának feltérképezésére. Azonban ez a módszer sokszor technikai nehézségekbe ütközik. A pusztán manuális elemzés túl sok időt és emberi erőforrást vesz igénybe. Automatikus detektálás esetén létre kell hozni egy alkalmas modellt, ami sokszor nem könnyű feladat. Gépi tanuló algoritmusok használatához rengeteg, emberek által felcímkézett adatra van szükség, ráadásul sokszor a betanított modellek nem elég pontosak, és nagyon érzékenyek az új környezetből származó adatokra. A madárhang detektáló kihívás ezen problémák megoldására invitálja a mély tanulás kutatóit.

A résztvevők feladata olyan algoritmusok készítése volt, amelyek egy körülbelül 10 másodperces felvételtől képesek eldönteni, hogy tartalmaz-e madárhangot vagy nem, a madár fajtájától függetlenül. Az algoritmusok többsége konvolúciós neurális hálókat alkalmaz spektrogram bemenetekkel. Én a győztes csapat (*Bulbul*) megoldását tervezem reprodukálni az eredményeikről publikált cikkük alapján². A Bulbul csapat egy előrecsatolt, vagyis elágazásoktól mentes konvolúciós neurális hálót alkalmazott, melynek bemenete mel skálájú, logaritmikus magnitúdójú spektrogramokat várt.

3.1. Az adatok

3.1.1. Források

A madárhang detektáló kihívás három különböző forrásból származó adathalmazzal biztosít. Az első a freefield1010¹⁴ projekt terepi felvételei, a FreeSound¹⁵ online adatbázisból származó felvételek kivonatainak gyűjteménye, melyek lokációban és környezetben igen változatosak. A második a Warblr¹⁶ tömegforrású, körülbelül 10 másodperces okostelefonos felvételeit tartalmazó adatbázis. A felvételek szélesen lefedik az Egyesült Királyság helyszíneit és környezetét, tartalmazzak időjárési és közlekedési hangokat, beszédet és még emberek általi madár imitációkat is. A harmadik adatkészlet a TREE kutatási projektből származik, amely felügyelet nélküli, távoli monitorozású felvevő berendezéseket telepített a csernobili tiltott zónába (Chernobyl Exclusion Zone). Az innen származó felvételek tartalmazzák különféle madarak és nagyobb emlősök hangjait, illetve az időjárás és rovarok zaját. A három adatgyűjtemény letölthető a kihívás weboldaláról¹⁷.

3.1.2. Az adatok struktúrája

A tanításhoz szükséges adatokat a freefield1010 (7690 felvétel) és a Warblr (8000 felvétel) adatgyűjteményekből nyerjük, ezekhez a felvételeken kívül rendelkezésre áll azok manuális osztályozása is. A felvételek címkéi vagy 1, vagy 0 értékűek aszerint, hogy szerepel-e a felvételen madárhang, vagy nem (1: a felvétel tartalmaz madárhangot, 0: nem tartalmaz). A freefield1010 összes felvétele pontosan 10 másodperc hosszú, a Warblr felvételei azonban változó hosszúságúak, a leghosszabb 22, a leg-rövidebb pedig 1 másodperces, többsége azonban szintén 10 másodperc körüli. A freefield1010 többnyire negatív címkéket tartalmaz (25%-ban van jelen madárhang), míg a Warblr legnagyobb része pozitívakat (76%-ban van jelen madárhang).

A teszteléshez használandó felvételek főleg a csernobili, kisebb részben pedig a Warblr adatgyűjteményből származnak (összesen 8620 minta). A teszteléshez szükséges címkéket nem tették nyilvánossá a verseny szervezői, csak a felvételeket. A verseny résztvevői elküldhették a tesztalmaz felvételeire kapott osztályozási eredményüket a versenybizottságnak, akik a náluk lévő címkék alapján elvégezték az eredmények kiértékelését. A verseny lezárta előtt, naponta egyszer lehetett a csernobili adatokra tesztelni, a kiértékeléshez az összes felvétel 15%-át használták fel (1293 felvétel). A verseny végén a teljes tesztalmazra elvégezték a modellek osztályozásának kiértékelését, ez a legjobb megoldásoknál mindössze pár ezrelék eltérést okozott a korábbi eredményekhez képest.

Összességében az adatstuktúra nagy kihívást jelent a felügyelt gépi tanulás alapú megközelítések számára. Az egyik tanulási forrásból főleg pozitív, míg a másiktól többnyire negatív minták állnak rendelkezésre, a két tanító halmaz különböző tulajdonságokkal rendelkezik, a teszt halmaz pedig egy harmadik, különböző forrásból lett vételezve.

3.1.3. A tesztalmaz manuális címkézése

A tesztalmazból számomra kizárólag a felvételek álltak rendelkezésre, ennél fogva szükségessé vált, hogy a tesztalmaz egy részét (összesen 2000 elemet, ami jóval több, mint a tesztalmaz korábban elmített 15%-a) manuálisan osztályozzam. Ez a 2000 elem már reprezentatívan képviseli a tesztalmaz egészét, várhatóan 1-2 ezrelék eltérést okozhat az összes elemre teszteléshez képest. A felvételek címkézéséhez írtam egy programot, ami megkönnyítette a munkát. A program véletlenszerűen kiválasztott 2000 felvételt, majd egyesével elkezdte lejátszani őket. Minden felvételt többször is meg lehetett hallgatni, a címkézést pedig bármikor meg lehetett szakítani. A program minden felvétel lejátszása alatt választ várt a felhasználótól: ha a válasz '0', vagy '1' volt, akkor a címke rögzítésre került egy .csv fájlban a felvétel azonosítójával, ha a válasz egy 'f' betű volt, akkor a program befejezte a futást,

következő megnyitáskor pedig ott folytatta, ahol előzőleg abbahagyta.

A manuális osztályozás nehéz feladatnak bizonyult. A felvételek jelentős részén nagyon hangos az alapzaj, ami minden más hangot nehezen kivehetővé tesz. A rovarok hangjai (például tücsökciripelés, kabócák hangja) sokszor nagyon nehezen különböztethető meg a madarak hangjától. Az egészen halk, rövid magas hangokról volt a legnehezebb megállapítani, hogy madarakhoz tartoznak-e, vagy valami más okozza őket. Pár emberi imitáció is nagyon meggyőzőre sikerült. Így összességében az általam címkézett teszhalmaz segítségével az állapítható meg, hogy mennyire hasonlóan osztályoz a gépi tanuló modell hozzám képest.

3.2. A felvételek feldolgozása

A neurális háló mel spektrogramokat használ, mint bemenet. A mel spektrogramok kiszámításához először spektrogramokat kell készítenünk a felvételekből, amit az ablakozott Fourier-transzformációval (short time Fourier-transform, STFT) tehetünk meg. Az STFT az időtérbeli diszkrét, mintavételi frekvenciával (f_s) jellemezhető jelet kisebb, egymással gyakran átfedő szakaszokra bontja, és ezeken a kisebb szakaszokon végez diszkrét Fourier-transzformációt.

3.2.1. Diszkrét Fourier-transzformáció (DFT)

A DFT együtthatóit a következő módon számolhatjuk:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N},$$

ahol x_n a transzformálandó adatsor n -edik eleme, N pedig az adatsor elemeinek száma. A DFT frekvenciakomponenseinek ($k \cdot \Delta f$) véges a felbontása. A legkisebb frekvencia (az alap frekvencia):

$$f_1 = \Delta f = \frac{1}{T} = \frac{f_s}{N},$$

ilyen távolságokra vannak egymástól a frekvenciakomponensek. Az együtthatók N komponenstől ismétlődnek (körbeérünk a komplex körön). Az eddigiekből következik, hogy a DFT az X_k frekvenciakomponensekre ($k \in [0, N-1]$) olyan együtthatókat számol, melyekkel jó közelítéssel visszakaphatjuk az eredeti jelünket a következő transzformációval, mint a komponensek szinuszoid függvényekként alkalmazott lineáris kombinációja:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{i2\pi kn/N}.$$

Ez a mostani probléma esetén azt jelenti, hogy a DFT-vel meghatározhatjuk, hogy a felvétel egy részén milyen frekvenciájú hangok szólaltak meg. Ha a jel elemszáma páros, $N/2$ -tól $N - 1$ -ig, ha páratlan, $(N - 1)/2$ -tól $N - 1$ -ig a negatív frekvenciák szerepelnek növekvő (pozitív irányba tartó) sorrendben. Valós jelek esetén a frekvenciakomponensek együtthatói hermitikusan szimmetrikusak:

$$X_k = X_{-k}^*,$$

ahol a $*$ a komplex konjugálást jelenti. Ez azt eredményezi, hogy valós, páros elemű jelek esetén elég csak $N/2$, páratlan elemű jeleknél pedig csak $(N - 1)/2$ frekvenciakomponenst kiszámolni, a további elemek redundánsak lesznek. DFT esetén a legmagasabb lehetséges frekvenciakomponens a Nyquist frekvencia, ami a mintavételi frekvencia fele ($f_s/2$). A DFT gyors számítási módja az FFT (Fast Fourier Transform) algoritmus, mely rekurzív módon addig osztja ketté a jelet, amíg az lehetséges, vagyis amíg a jel páros elemszámú. Ezt követően minden kisebb részletre kiszámolja a DFT-t, melyeket fel tud használni az egész jel transzformációjának kiszámításához, azonban megspórolva rengeteg ismétlődő lépést. A DFT futási ideje $O(N^2)$, míg az FFT futási ideje csak $O(N \log(N))$.

3.2.2. Rövid idejű Fourier-transzformáció (STFT)

Az STFT kiszámításához szükségünk van egy ablakméretre (`n_fft`), mely megadja, hogy hány elemet tartalmaz az a felvételrészlet, amire az FFT-t elvégezzük és egy ugrásméretre (`hop_length`), ami megadja a felvételen végzett FFT-k középpontjainak távolságát. Az eljárás során az algoritmus a jel elejét és végét feltölti a jel széleinek tükörképével (`padding='reflect'`) úgy, hogy a megadott időpontokban végzett Fourier transzformációk az ablakok középpontjaira essenek. A jlrészletekre ezenkívül még egy Hann ablakfüggvényt is alkalmaztam, ami lecsengeti az ablak széleit, így az átfedő részek kevésbé számítanak bele a Fourier transzformációba és simább spektrogramot kaphatok. A Hann ablakfüggvény képlete:

$$W(x) := \begin{cases} \frac{1}{2} \left(1 + \cos \left(\frac{2\pi x}{L} \right) \right) = \cos^2 \left(\frac{\pi x}{L} \right), & \text{ha } |x| \leq L/2 \\ 0, & \text{ha } |x| > L/2 \end{cases}$$

A spektrogramot az STFT magnitúdóinak ábrázolásával kapjuk, mely során a fázist elhagyjuk és csak a komplex együtthatók abszolútértékét tartjuk meg. Az eredmény egy kétdimenziós mátrix, sorai a frekvenciákat, oszlopai pedig az időpontokat reprezentálják.

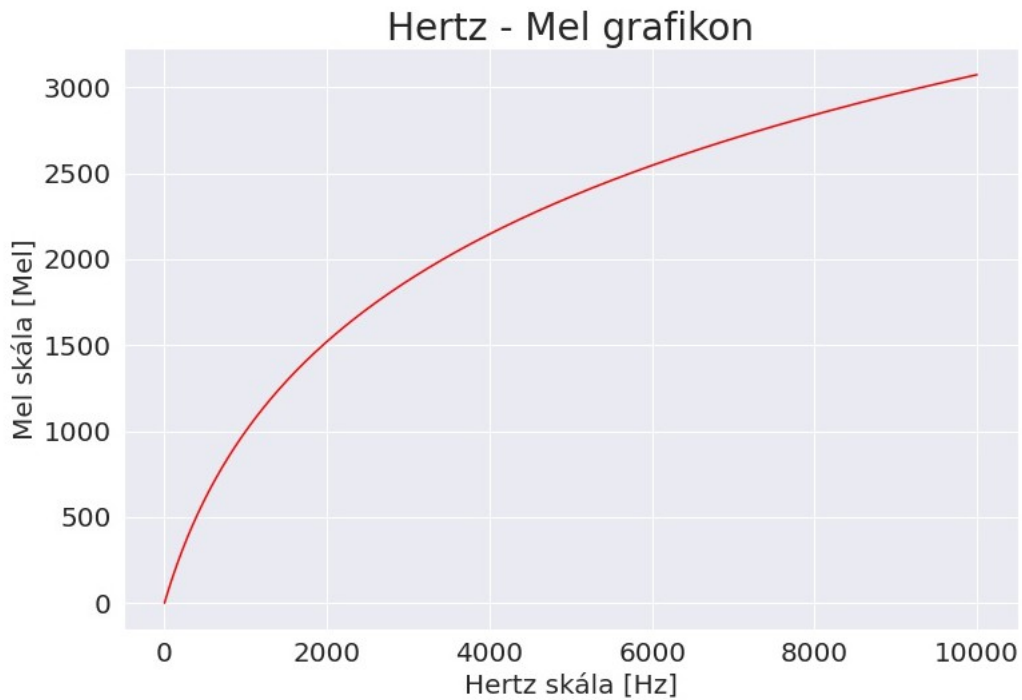
3.2.3. Mel spektrogramok

A mel spektrogram elkészítéséhez ezután a spektrogram frekvenciatengelyét át kell konvertálni a mel skálára. A mel (melody) skála a hangmagasságok emberek által egyenlőnek érzékelt távolságai. A frekvenciaskáláról mel skálára áttérés formulája és inverze itt látható:

$$m = 2595 \log_{10} \left(1 + \frac{f}{700} \right) = 1127 \log \left(1 + \frac{f}{700} \right),$$

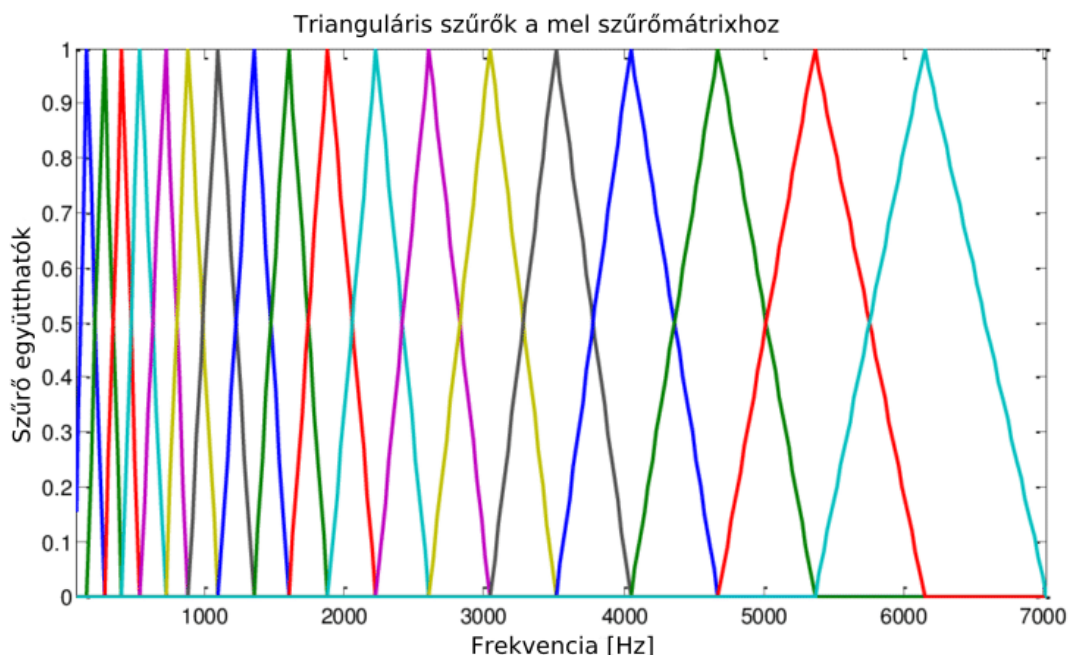
$$f = 700(10^{m/2595} - 1) = 700(e^{m/1127} - 1),$$

a két skála közös grafikonja pedig a 9. ábrán tekinthető meg.



9. ábra. Mel skála a frekvencia függvényében.

A mel skálára transzformáláshoz el kell készíteni a mel szűrőmátrixot (mel filter bank). Ehhez első lépésben ki kell választanunk a legmagasabb és legalacsonyabb frekvenciát, amelyek közötti frekvenciaértékekre szükségünk lesz (ezek önkényesen megválasztható hiperparaméterek). Ezt a két frekvenciát át kell konvertálnunk a mel skálára a fenti formulával, majd a két mel érték között egymástól egyenletes távolságra felvenni a választott mennyiségű mel pontokat (a mel pontok száma szintén hiperparaméter). A mel értékeket az inverz formulával visszaalakítjuk Hertz-re (ekkor a pontok már logaritmikus skálán helyezkednek el). A kapott pontokat az eredeti frekvenciákhoz legközelebbi értékekre kerekítjük, hogy megőrizzük az eredeti felbontásból származó pozíciókat. Ezt követően trianguláris filtereket kell készítenünk, melyeket a 10. ábra szemléltet.



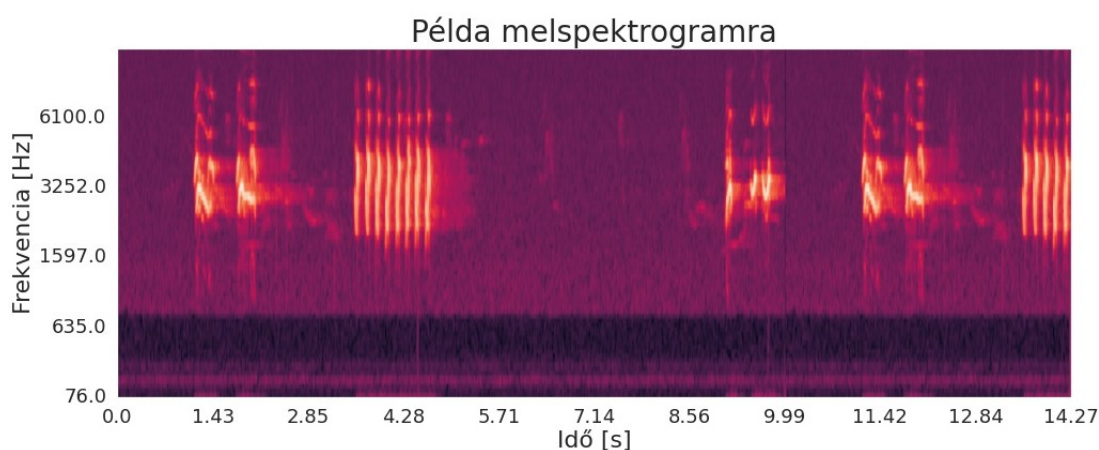
10. ábra. Egy lehetséges példa a trianguláris szűrők konfigurációjára.¹⁸ Minden háromszög csúcsait három egymást követő (balról jobbra) mel pont adja. A két szélső csúcson kívül eső frekvenciapontok súlya zérus lesz, a belül eső frekvencia pontok a háromszög egyeneseinek megfelelő súlyértékeket kapnak.

Minden filter középpontja egy mel pont, szélei pedig a középpontok szomszédos mel pozíciói (itt a mel értékek már át lettek váltva Hertz-re). A szűrő középpontjának értéke 1, a többi pont a széleig a háromszög egyenesein helyezkedik el, a széleken túl pedig minden érték 0. A szűrők külön vektorokat, együtt pedig egy kétdimenziós mátrixot alkotnak, minden sor egy külön szűrőt. Ezzel a mátrixszal beszorozva a spektrogramot az összes időpont frekvenciakomponens együtthatóira elvégezhető a mel együtthatókra történő transzformáció, mellyel elkészül a mel spektrogram. A mel spektrogramok egyik fő alkalmazási területe a beszéd-szintézis.¹⁹

3.2.4. Mel spektrogramok megvalósítása

Bulbul csapata 1024 elemű ablakméretet, 315-ös ugrásméretet és 22050 Hz -es mintavételezési frekvenciát használt az STFT-k elkészítéséhez, majd egy 80 elem hosszú, 50 Hz és 11000 Hz között elhelyezkedő mel szűrőmátrixszal átkonvertálta a spektrogramjait mel spektrogramokra, és a magnitúdók természetes logaritmusát vette. A mel spektrogramok értékei frekvenciakomponensenként standardizálva lettek zérus átlagra és egységnyi szórásra. Ezt egy köteg normalizációs réteggel oldották meg a hálózat struktúrájának elején. Ezen kívül még alkalmaztak egy mel spektrogramonkénti (vagyis nem a sokaságra, hanem az egyes spektrogramokra vonatkozó) átlagolást is, minden mel spektrogramból levonták az időbeli átlagaikat, ezzel eltávolítva a frekvenciafüggő (színes) zajokat.

Én a mel spektrogramok elkészítéséhez a Python Librosa nevű könyvtárát használtam. Egy saját algoritmussal teszteltem a könyvtár mel spektrogram készítőjének működését, és azt tapasztaltam, hogy a programok megfelelően működnek, egymáshoz nagyon hasonló eredményeket kaptam a saját és a beépített verzióval. Így nyugodt szívvel felhasználtam a könyvtár jól optimalizált függvényeit és írtam egy programot, ami elvégzi a felvételek transzformációját. Ehhez először meg kellett vizsgálnom a felvételek hosszait és címkéit. Mivel adott, hogy milyen széles lehet egy spektrogram (időben nézve körülbelül 14 másodperc), ki kellettennem a tanítóhalmazból minden olyan felvételt, ami hosszabb, mint a megadott keret és pozitívan van felcímkézve (mivel így egy vágás el is tüntethetné a címkét befolyásoló madárhangot). Összesen két ilyen elem volt a Warblr adathalmazból. A 14 másodpercnél hosszabb, de negatív címkéjű felvételeket nyugodtan megrövidítettem, a rövidebbeket pedig úgynevezett loopolással meghosszabbítottam. A loopolás annyit jelent, hogy a felvétel végét addig bővítem a felvétel elejétől történő ismétlésével, amíg el nem érem a kívánt hosszt. A loopolás tetszőleges mennyiségben végezhető, amíg csak szükséges. A loopolások kezdeténél szakadások keletkeznek a hirtelen hangváltások miatt. Ezt azzal orvosoltam, hogy loopolás esetén rövid exponenciális lecsengést alkalmaztam a felvétel végén és ismétlődő elején. A felvétel így nem megszakadt, hanem lehalkult és felhangosodott, azonban ennek ideje jóval rövidebb volt, mint egy potenciális madárhang, kevesebb, mint egy század másodperc. A programomnak mindösszesen egy forrásmappa útvonalát, mely tartalmazza a .wav kiterjesztésű felvételeket és egy célmappát kellett megadni argumentumként, melybe az elkészült mel spektrogramokat hdf5 tömörítéssel elmentheti. A 11. ábrán példaként az egyik kész melspektrogram tekinthető meg.



11. ábra. A freefield1010 adatgyűjtemény 71838 azonosítószámú, pozitív címkéjű felvételének melspektrogramja. Tíz másodpercnél a vékony függőleges fekete vonal mutatja a loopolás kezdetét. A világospiros mintázatok a felvételen szereplő madárhangok.

3.3. A hálózat felépítése

A legjobb eredményt elérő csapat egy 1000 elem hosszú (14 másodperc), széles receptív mezővel rendelkező konvolúciós, előrecsatolt neurális hálót alkalmazott, melynek végső kimenete egyetlen bináris neuron, mely 0 és 1 közötti folytonos értékeket tud képezni. Ahogy az a függelék 2. táblázatában is látható, a hálózat első része négy egymást követő konvolúciós és maxpooling rétegből áll. Az eredeti megoldás $1 \times 1000 \times 80$ -as bemenetéhez képest én $1 \times 80 \times 1000$ -es bemeneteket használtam, mert könnyebb volt elképzelni a spektrogramot 80×1000 -es mátrixként és egyszerűbben tudtam ábrázolni (így nem volt szükséges a spektrogram elforgatása az ábrázolás előtt). Ennek szerencsére nincs jelentősége a modell pontosságának szempontjából, a hálózat eredetileg nem tudja, hogy milyen irányból kellene néznie az adatokat, ugyanúgy rá tud tanulni, mint az eredeti esetben. Az egyetlen extra módosítás, amit emiatt el kellett végezni, az az első és második 3×3 -as konvolúciós kernelek után a harmadik és negyedik, eredetileg 3×1 -es kernelek 1×3 -assá alakítása. A konvolúciós és pooling rétegek végére a kimenet dimenziója $16 \times 8 \times 11$ -re redukálódott, melyből egy dimenzió csökkentő réteg (Flatten) vektort készít, és azt adja tovább a bemenet osztályozását végző teljesen összekapcsolt hálózatnak. A teljesen összekapcsolt hálózat három rétegből áll, neuronjaik száma rendre 256, 32 és 1. Az aktivációs függvények, leszámítva a legutolsó rétegnél használt szigmoid aktivációs függvényt, minden konvolúciós és teljesen összekapcsolt rétegre Leaky ReLU-k voltak $\alpha = 0.01$ paraméterrel. A hálózat összes tanítható paramétere 373009 volt.

3.4. A hálózat tanítása

A tanítást az eredeti 64, majd később pedig kísérletképpen 32 elemű kötegelt gradiens ereszkedéssel végeztem az ADAM optimalizáló használatával. A tanulási rátát 0.001-nek választottam, és 1500 epochonként, összesen kétszer, tízzel leosztva redukáltam az értékét. Így összesen 4500 epochig tanítottam egy modellt, ami jelentősen eltér az eredeti megoldástól. Az eredeti megoldásban a tanulási rátát csak akkor frissítették, ha háromszor egymásutáni 1500 epoch alatt nem volt fejlődés a veszteségfüggvényben. A harmadik ilyen eset után a tanulás befejeződött (tanulási ráta ejtés és korai megszakítás módszere). Ehhez azonban a győztes csapat leírása alapján megközelítőleg 80000 epoch futtatására lett volna szükség egy modell esetében. Ez sajnos beláthatatlan időt vett volna igénybe, ezért döntöttem 4500 epoch mellett, ami még mindig meglehetősen soknak számít (közepes erősségű P4 GPU mellett körülbelül 24 óra, de az eredeti ötszörös keresztvalidációval már öt napba telik egy teljes tanítás).

Mindegyik teljesen összekapcsolt réteg előtt egy-egy 50%-os dropout réteg lett elhelyezve, mely segíti a hálózat regularizációját. Annak, hogy a madárhang a

felvétel mely részén szólal meg, nincs jelentősége, illetve a madárhangok magassága is változhat, ezért a tanítás során adatszövelést alkalmazok. A spektrogramokat egy véletlenszerű számmal, periodikusan eltolom az időtengely mentén, ezenkívül pedig a mel skálán alkalmazok ± 1 mel sávon belüli lineárisan interpolált eltolást a hangmagasságok véletlenszerű megváltoztatásához.

3.5. Keretrendszer és futtatás

A program megírásához a Google Tensorflow 2.0 keretrendszert használtam. A Tensorflow Keras Sequential modellje egyszerűvé tette a modell elkészítését. Az adatszövelő és a spektrogramonkénti, zérus időbeli átlagoló rétegeket úgynevezett custom layerek segítségével készítettem el. A custom layerek a Tensorflow Keras layers Layer osztályából származtathatók, a Tensorflowban nem implementált, egyedi rétegek létrehozására használják. A Sequential modell úgynevezett szimbolikus modellt alkot. A szimbolikus modellek legegyszerűbben gráfokként képzelhetők el, melyek szimbolikus tenzorokat kezelnek. A szimbolikus tenzorok olyan tenzorok, amelyek még nem tárolnak semmilyen értéket, így a Sequential modell nem hoz létre tényleges tenzorokat, csak a műveletek folyamatát tárolja el. Ilyen tenzorok kezelésére használhatóak a Tensorflow Tensor objektumai és függvénykönyvtára. A NumPy modulhoz hasonlóan, A Tensorflow könyvtárában a legtöbb hasznos művelet és tenzor kezelő eljárás elérhető.

A tanulás futtatásához az egyetem belső szervergépeit használtam, majd később a Google Cloud Platform virtuális gépeit P4 GPU-kal.

3.6. Validáció és tesztelés

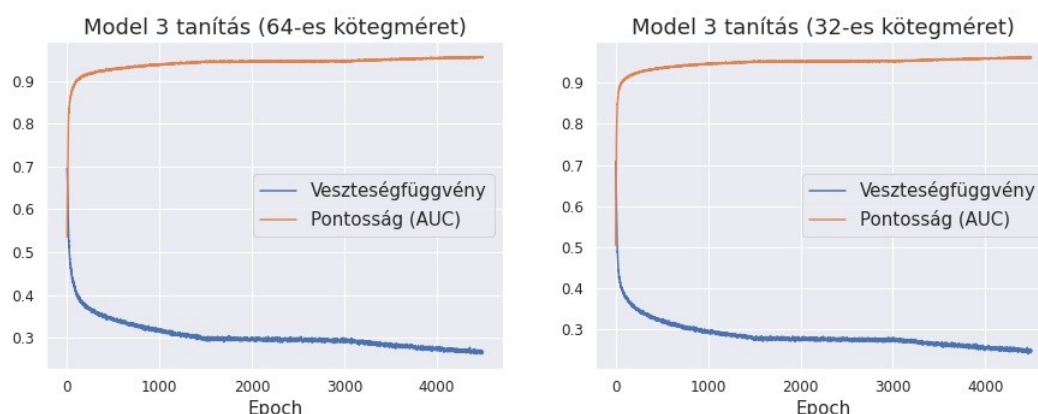
A modellem pontosságának mérésére ROC görbe alatti területet használtam (AUC), a validációra pedig ötszörös keresztvalidációt. A tesztelést az általam címkézett tesztalmaz felvételein végeztem, a validáció során tanított öt modell sokaságát alkalmazva. A modellek külön predikcióinak átlagát vettem eredményül, a predikciókra pedig a címkék segítségével kiszámoltam az AUC mértéket.

A keresztvalidációk és tesztelések eredményei		
Kötegméret	64	32
validáció model 1	0.9488	0.9462
validáció model 2	0.9442	0.9470
validáció model 3	0.9579	0.9574
validáció model 4	0.9500	0.9549
validáció model 5	0.9487	0.9509
validációk átlaga	0.9499	0.9513
tesztelés eredménye	0.8775	0.8836

1. táblázat. A validációk és teszteredmények összehasonlításához készült táblázat.

3.7. Az eredmények értékelése

Mind a 64-es, mind a 32-es kötegméret esetén pár százalékkal rosszabb eredményt kaptam a validációkra, mint az eredeti megoldásban szereplő 0.975 átlagérték, azonban a rövidebb tanulási idő magyarázatot adhat az eltérésekre. A következő, 12. ábrán jól láthatóak, hogy a harmadik modell tanulóhalmazhoz tartozó veszteségfüggvényei még csökkenő tendenciát mutattak, tehát a leállítások a teljes tanuláshoz szükséges idő előtt mentek végbe. Hosszabb tanítás esetén jobb validációs eredményeket kaphatnék. A többi modell tanulási görbéi megtekinthetők a függelékben.



12. ábra. A 64-es és 32-es kötegméretre végzett validációk tanítóhalmazra vonatkozó grafikonjai. Jól látható, hogy a veszteségfüggvények a tanítás végén még erősen csökkenő tendenciát mutatnak.

A validációk átlagai alig térnek el egymástól a két kötegméret esetében (a 32-es kötegméret egy hajszállal jobb eredményt adott), azonban a tesztelésnél a 32-es kötegméret mellett tanított modell már jelentős mértékben jobban teljesített, mint a 64-es kötegméretű verzió. Ezt feltehetőleg az okozza, hogy a 32-es kötegméret mellett a modell súlyainak frissítése zajosabban konvergál a minimumok felé. Ez a viselkedés általában azt eredményezi, hogy a modell jobban generalizál, vagyis ismeretlen, a tanítóhalmaztól eltérő tulajdonságú adatokra is jobban fog tudni teljesíteni.²⁰ Nekünk a mostani kereszttanítás esetén pont erre van szükségünk. A 32-es kötegmérettel kapott tesztelési eredmény már jó közelítése a verseny győztes megoldásának, a rövid tanítási idő ellenére is.

4. Diszkusszió

A szakdolgozatom során az eredeti témámtól elkanyarodva madárhangok mély tanulás segítségével történő detektálásával foglalkoztam.

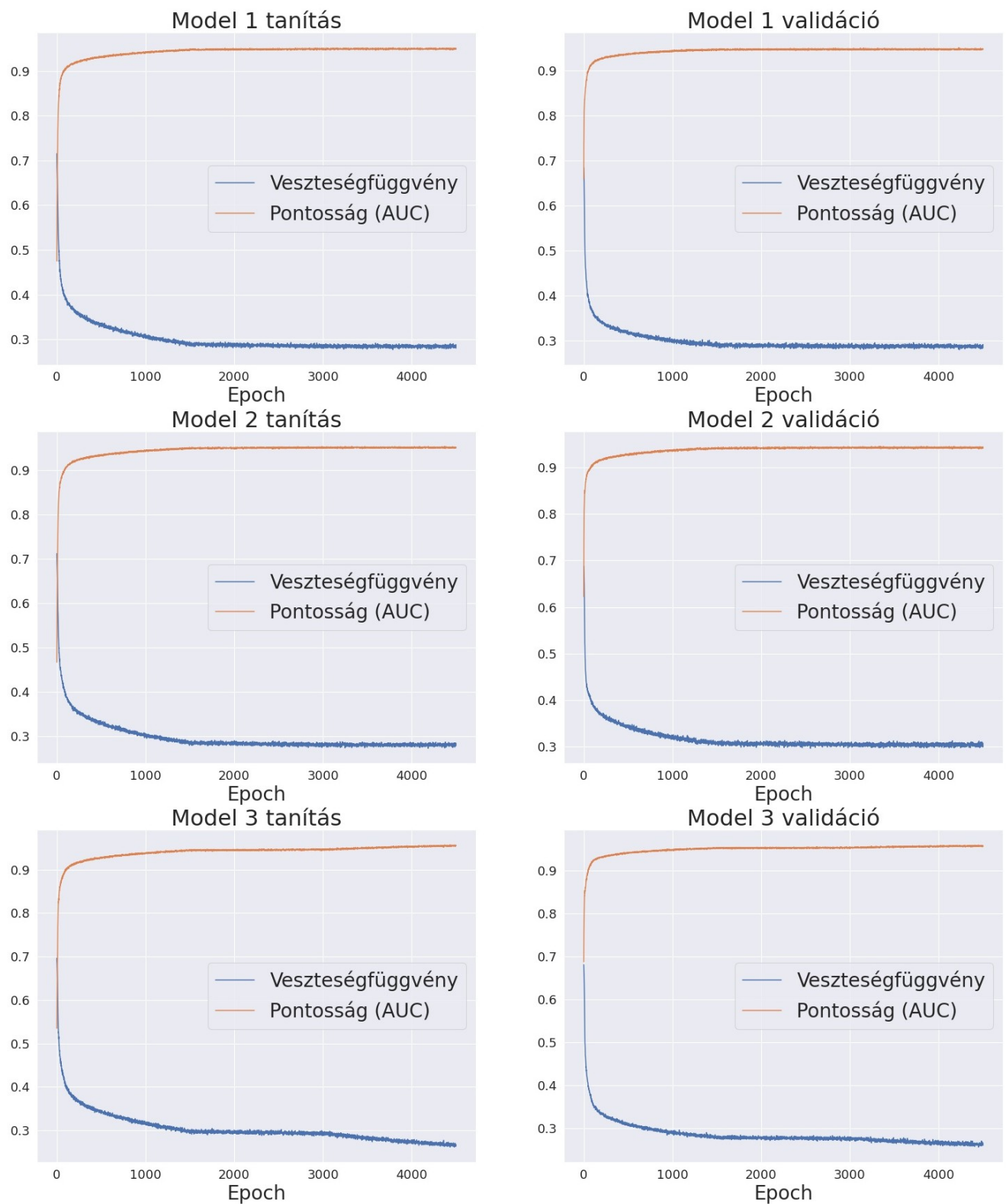
Munkám során megismerkedtem a mély neuronhálók matematikai alapjaival, a különböző hálóstruktúrákkal, a tanítás lépéseivel, a hálózat regularizációjával és még sok más gépi tanulás alapú módszerrel. Az adatok feldolgozása során tanultam normalizációs és standardizációs technikákról, illetve mélyebben belemerültem a mel spektrogramok elkészítéséhez szükséges elméletbe, amely magába foglalta a Fourier-transzformáció típusainak (FT, DFT, FFT) és egyes felhasználási módjainak (például STFT spektrogramok készítéséhez) mélyebb megértését.

Technikai oldalról közelítve gyakorlatot szereztem a Tensorflow keretrendszer használatában, megismertem és teszteltem a librosa könyvtár működését és beletanultam a Google Cloud Platform virtuális gépeinek GPU-val történő használatába távoli elérés segítségével (ssh).

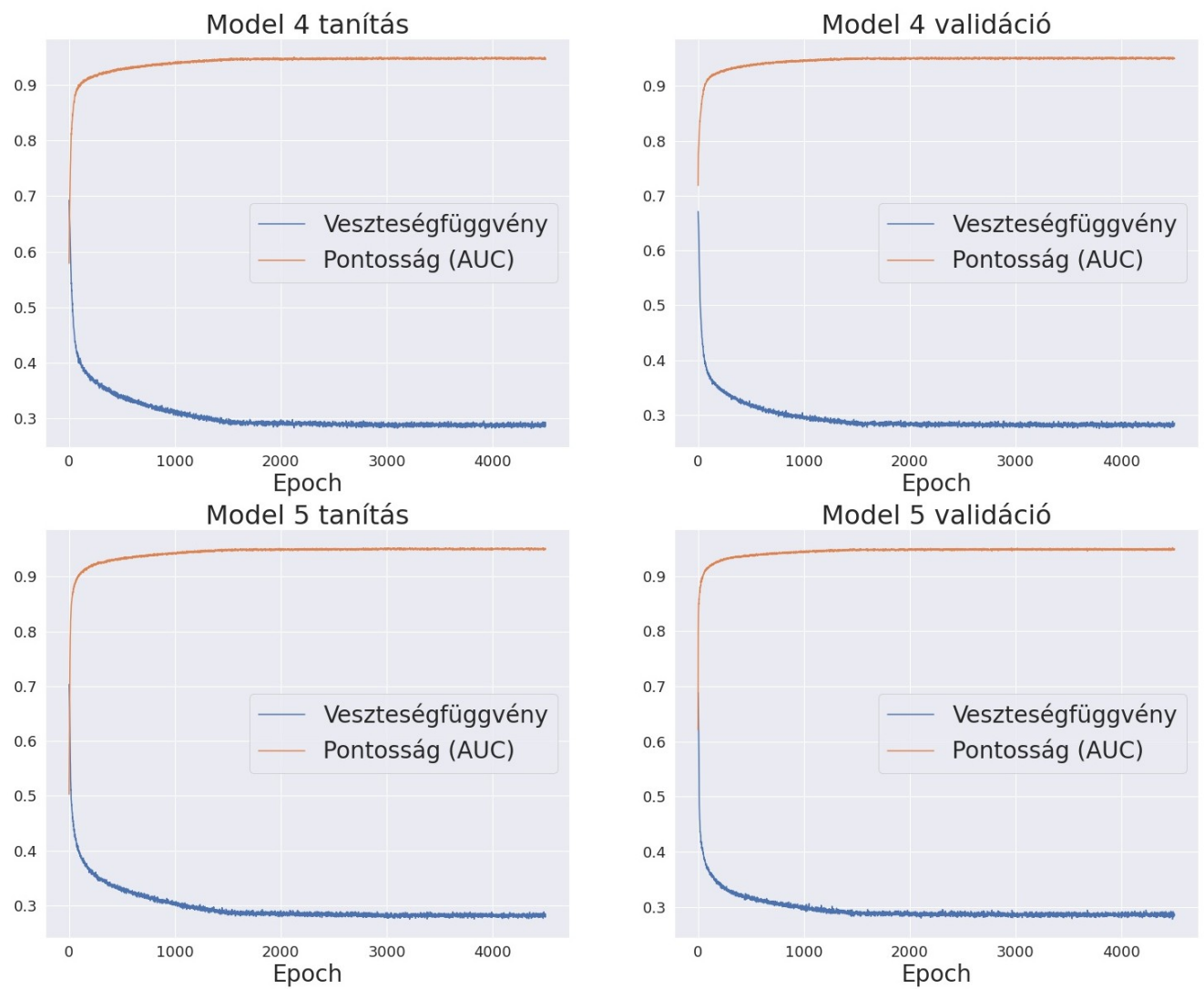
Létrehoztam egy felügyelt tanulás alapú, előrecsatolt konvolúciós neurális hálót a Bird Audio Detection Challenge mély tanulási verseny győztes megoldásának mintájára. Kipróbáltam az eredeti 64-es kötegméret mellett egy 32-es verziót is a tanítás során, mellyel vizsgáltam a kisebb kötegméretek generalizáló hatását. A 32-es kötegméretre az ötszörös keresztvalidáció átlagos értéke 0.9513 AUC lett, a teszthalmazon pedig 0.8836 AUC pontosságot értem el vele. Ezzel sikerült megközelítenem az eredeti megoldás végeredményét.

5. Függelék

5.1. Keresztvalidáció 64-es kötegméretre

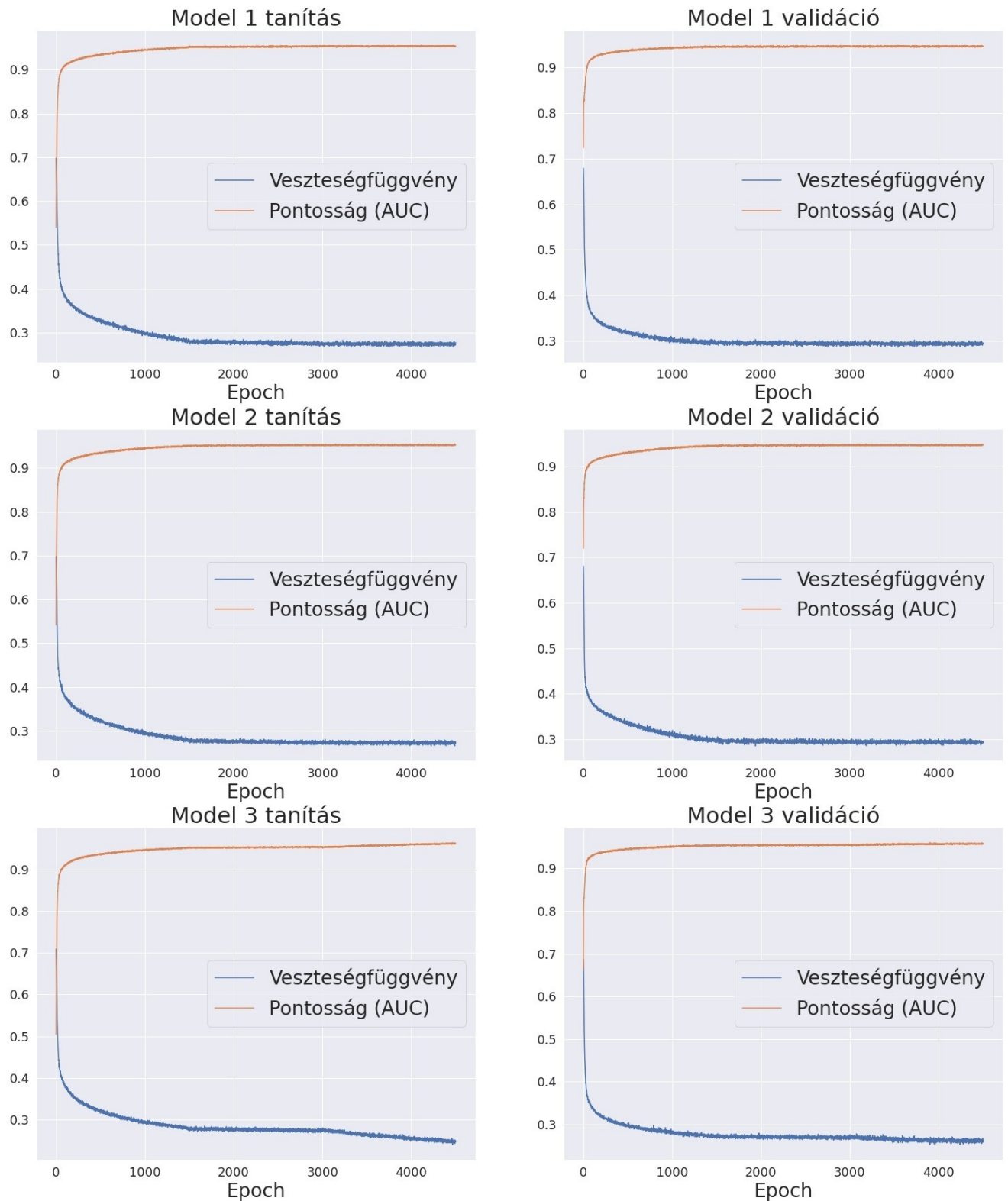


13. ábra. A 64-es kötegméretű keresztvalidáció első három tanulási és validációs grafionja az epochok függvényében. A harmadik modellen jól látszik, hogy idő előtt lett leállítva a tanulás.

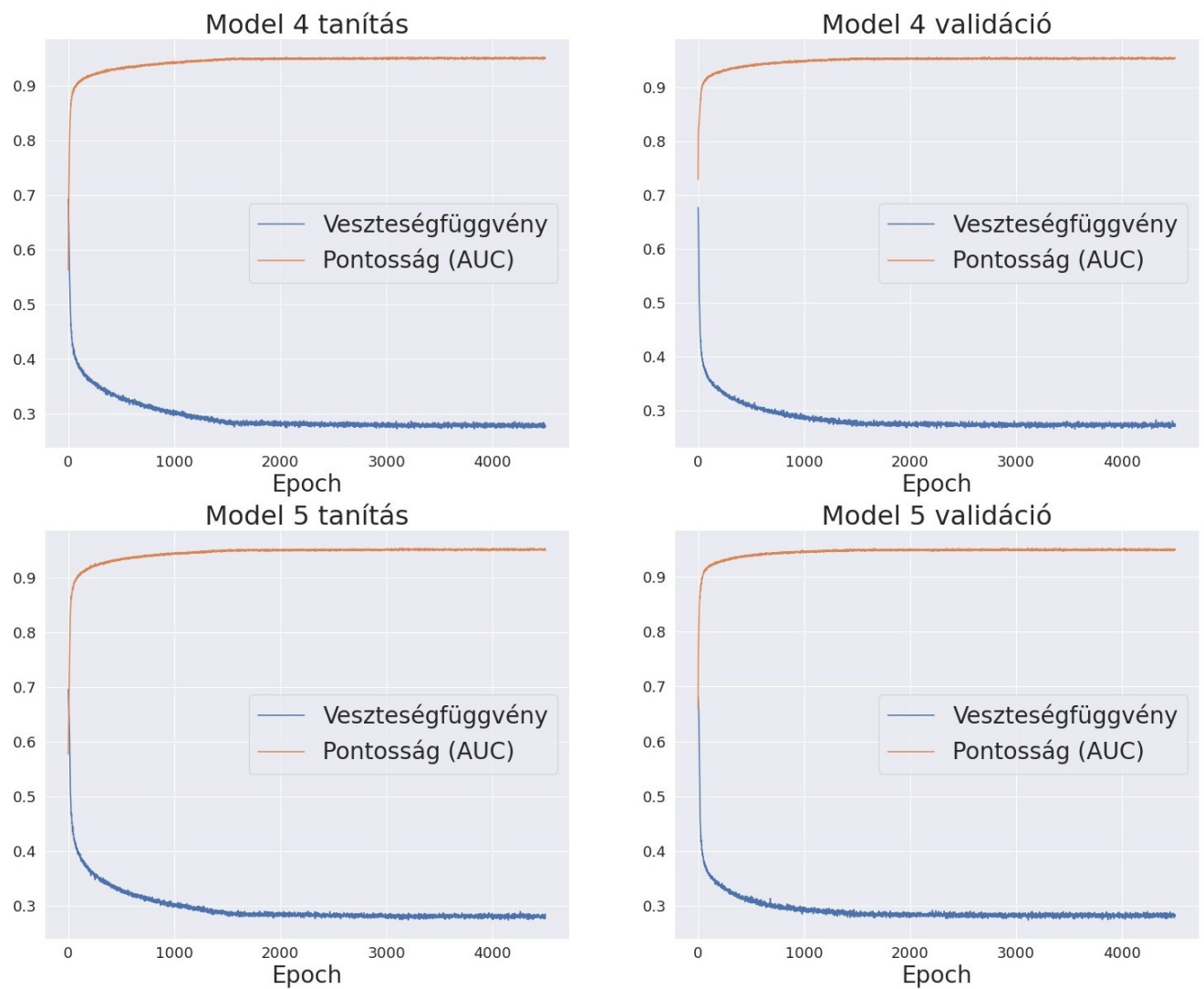


14. ábra. A 64-es kötegméretű keresztvalidáció negyedik és ötödik tanulási és validációs grafionja az epochok függvényében.

5.2. Keresztvalidáció 32-es kötegméretre



15. ábra. A 32-es kötegméretű keresztvalidáció első három tanulási és validációs grafionja az epochok függvényében. A harmadik modellen jól látszik, hogy idő előtt lett leállítva a tanulás.



16. ábra. A 32-es kötegméretű keresztvalidáció negyedik és ötödik tanulási és validációs grafionja az epochok függvényében.

5.3. A neurális hálózat felépítése

Architektúra	
Bemenet	$1 \times 80 \times 1000$
Konvolúciós(3×3)	$16 \times 78 \times 998$
Pool(3×3)	$16 \times 26 \times 332$
Konvolúciós(3×3)	$16 \times 24 \times 330$
Pool(3×3)	$16 \times 8 \times 110$
Konvolúciós(1×3)	$16 \times 8 \times 108$
Pool(1×3)	$16 \times 8 \times 36$
Konvolúciós(1×3)	$16 \times 8 \times 34$
Pool(1×3)	$16 \times 8 \times 11$
Teljesen összekapcsolt	256
Teljesen összekapcsolt	32
Teljesen összekapcsolt	1

2. táblázat. A madárhangok detektálásához használt konvolúciós neurális hálózat struktúrája.

Hivatkozások

- [1] D. Stowell, M. D. Wood, H. Pamuła, Y. Stylianou, and H. Glotin, „Automatic acoustic detection of birds through deep learning: The first Bird Audio Detection challenge,” *Methods in Ecology and Evolution*, vol. 10, no. 3, pp. 368–380, 2018.
- [2] T. Grill and J. Schluter, „Two convolutional neural networks for bird detection in audio signals,” in *2017 25th European Signal Processing Conference (EUSIPCO)*, IEEE, Aug. 2017.
- [3] W. S. McCulloch and W. Pitts, „A logical calculus of the ideas immanent in nervous activity,” *The Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, dec 1943.
- [4] F. Attneave, M. B., and D. O. Hebb, „The organization of behavior a neuropsychological theory,” *The American Journal of Psychology*, vol. 63, p. 633, oct 1950.
- [5] T. R. Halfhill, „The mythology of moore's law: Why such a widely misunderstood law’ is so captivating to so many,” *IEEE Solid-State Circuits Society Newsletter*, vol. 11, pp. 21–25, sep 2006.
- [6] Wikipédia, „Idegsejt.” <https://hu.wikipedia.org/wiki/Idegsejt>. Utoljára látogatva: 2021.05.22.
- [7] N. D. Santo, S. Deparis, and L. Pegolotti, „Data driven approximation of parametrized PDEs by reduced basis and neural networks,” *Journal of Computational Physics*, vol. 416, p. 109550, sep 2020.
- [8] Á. Fodor, A. Lőrincz, and L. Kopácsi, „Mély neuronhálók matematikájának alapjai (előadás jegyzet),” ELTE IK MSc 2020.
- [9] D. L. Tóth, „Mesterséges neuronhálók és alkalmazásaik (regularizáció, 5. diászor).” <https://www.inf.u-szeged.hu/~tothl/ann/05.%20Regulariz%C3%A1ci%C3%B3.pdf>, 2020. A tananyag az EFOP-3.5.1-16-2017-00004 pályázat támogatásával készült.
- [10] D. H. Hubel and T. N. Wiesel, „Receptive fields, binocular interaction and functional architecture in the cat's visual cortex,” *The Journal of Physiology*, vol. 160, pp. 106–154, jan 1962.
- [11] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, „Convolutional neural networks: an overview and application in radiology,” *Insights into Imaging*, vol. 9, pp. 611–629, jun 2018.

- [12] S. Gidaris and N. Komodakis, „Object detection via a multi-region and semantic segmentation-aware cnn model,” in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.
- [13] L. Zanella-Calzada, C. Galván-Tejada, N. Chávez-Lamas, J. Rivas-Gutierrez, R. Magallanes-Quintanar, J. Celaya-Padilla, J. Galván-Tejada, and H. Gamboa-Rosales, „Deep artificial neural networks for the diagnostic of caries using socioeconomic and nutritional features as determinants: Data from NHANES 2013–2014,” *Bioengineering*, vol. 5, p. 47, jun 2018.
- [14] D. Stowell and M. D. Plumbley, „An open dataset for research on audio field recording archives: freefield1010,” *CoRR*, vol. abs/1309.5275, 2013.
- [15] „Freesound database.” <http://freesound.org>. Utoljára látogatva: 2021.05.27.
- [16] „Warblr database.” <http://warblr.net>. Utoljára látogatva: 2021.05.27.
- [17] „Bird audio detection challenge.” <http://machine-listening.eecs.qmul.ac.uk/bird-audio-detection-challenge>. Utoljára látogatva: 2021.05.27.
- [18] M. Yusnita, M. Paulraj, S. Yaacob, R. Yusuf, and A. Shahrman, „Analysis of accent-sensitive words in multi-resolution mel-frequency cepstral coefficients for classification of accents in malaysian english,” *International Journal of Automotive and Mechanical Engineering*, vol. 7, pp. 1053–1073, jun 2013.
- [19] L. Juvela, B. Bollepalli, J. Yamagishi, and P. Alku, „Gelp: Gan-excited linear prediction for speech synthesis from mel-spectrogram,” 2019.
- [20] D. Masters and C. Luschi, „Revisiting small batch training for deep neural networks,” *CoRR*, vol. abs/1804.07612, 2018.

Köszönetnyilvánítás

Ezúton is szeretnék köszönetet mondani témavezetőmnek, dr. Stéger Józsefnek a rengeteg ötletért, a rám szakított hosszú órákért, a nehezebb időszakok megértéséért és végtelen türelméért. Továbbá szeretnék köszönetet mondani az én csodálatos páromnak a sok bátorításért és tanácsért, amivel a szakdolgozatom megírása alatt elhalmozott.

NYILATKOZAT

Név: Jakobi Ádám

ELTE Természettudományi Kar, szak: Fizika BSc

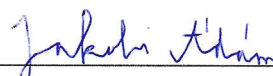
NEPTUN azonosító: JE9Z6Z

Szakdolgozat címe:

Örvös légykapók énekének gépi tanulás alapú osztályozása

A **szakdolgozat** szerzőjeként fegyelmi felelősségem tudatában kijelentem, hogy a dolgozatom önálló szellemi alkotásom, abban a hivatkozások és idézések standard szabályait következetesen alkalmaztam, mások által írt részeket a megfelelő idézés nélkül nem használtam fel.

Budapest, 2021.05.31.



a hallgató aláírása