TANSZÉKVEZETŐ

SZAKDOLGOZAT FELADAT

Osváth Barnabás
Mérnökinformatikus hallgató részére

Balkamrai-hipertrófia osztályozása MRI képek alapján

A mély neuráis hálók alkalmazása az élet minden területén egyre elterjedtebb. Kiemelkedő fontosságú az egészségügy, mint felhasználási terület. Az orvosi diagnosztikában egyre fontosabb szerepet töltenek be az automatizált, „machine learning" alapú technológiák. Ezen megoldásoknak nagy az adatigényük, de az elmúlt években hangsúlyossabbá vált az orvosi adatbázisok építése kutatási célokból.

A hallgató a félév során Városmajori Szív-és Érgyógyászati Klinikával közösen folytatott projektbe kapcsolódhat be, melynek célja a balkamrai hipertrófia diagnosztizálásának támogatása automatikus megoldásokkal. A feladat MRI képek osztályozása egészséges, HCM-es, illetve egyéb (EMF, Fábry-kór stb.) kategóriákba. A felhasználható adathalmazt a betegek és a kontrollcsoport „nyers", rövidtengelyi MRI képei alkotják. Az MRI képek, mint idősoros („sequential") adatok kerülnek felhasználásra.

A hallgató feladatának a következőkre kell kiterjednie:

- Végezzen irodalomkutatást orvosi képek idősoros elemzésével kapcsolatban
- Végezze el a felhasználandó adatok előfeldolgozását
- Tervezzen modellt a feladat elvégzésére, és módszert az eredmények kiértékelésére
- Implementálja a modellt és a kiértékelési módszert
- Helyezze kontextusba a kapott eredményeket

Tanszéki konzulens: Budai Ádám doktorandusz

Budapest, 2020. október 4.

Dr. Charaf Hassan
egyetemi tanár
tanszékvezető

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék
H-1117 Budapest, Magyar tudósok krt 2. Q.B.207
Telefon: 463-2870. Fax: 463-2871
www.aut.bme.hu

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar

Balkamrai-hipertrófia osztályozása MRI képek alapján

Szakdolgozat

Készítette
Osváth Barnabás

Konzulens
Budai Ádám

2020. december 10.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Osváth Barnabás*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

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

Budapest, 2020. december 10.

Osváth Barnabás
hallgató

Kivonat

A mély neurális hálókon alapuló technológiák egyre elterjedtebbek az élet minden területén. A 2010-es évek elején bekövetkező fejlődés a GPU-k teljesítményében a képfeldolgozás esetében is berobbantotta ezt a megközelítést. Különösen fontos szektor az egészségügy, hiszen a jól teljesítő megoldások megkönnyíthetik az orvosok dolgát, és akár emberi életeket is menthetnek. Ezek a megoldások főleg az orvosi vizsgálati folyamatokba integrálható automatikus diagnosztikai technológiák, amiket az tesz lehetővé, hogy egyre nagyobb orvosi felvételekből álló adatbázisok épülnek ki.

Szakdolgozatomban szívről készült MRI felvételeket klasszifikálok diagnózisok szerint, elsősorban balkamrai-hipertrófiát, majd azon belül hipertrófiás kardiomiopátiát (HCM). Kétféle osztályozást is alkalmazok, először egészséges és beteg, majd egészséges, HCM-es, és egyéb kategóriákba sorolom a felvételeket. Mindezt széleskörűen elterjedt, előtanított konvolúciós hálók finomhangolásával, és az általuk kinyert jellemzők visszacsatolt hálóba vezetésével kísérlem meg, így figyelembe véve a szív szerkezetét és időbeli mozgását egyaránt.

Az implementált modelleken hiperparaméter optimalizációt hajtok végre, majd a teljesítményüket kiértékelem néhány, osztályozási feladatok esetén általánosan használt metrikán, például tévesztési mátrixokon. Végül a metrikák alapján összehasonlítom a különböző konvolúciós háló architektúrák teljesítményét, illetve az egyik CNN felhasználásával néhány tanítási módszer hatását az eredményekre.

Végül a legjobb elért eredményeim a két osztályba sorolás esetén 90%-os teszt pontosság és F_1 érték, míg három osztály esetén 83%-os teszt pontosság és 84%-os F_1 érték.

Véleményem szerint a közeljövőben a mély neurális hálokon alapuló automatikus orvosi diagnosztikai eszközök fontos kiegészítései lesznek az orvosok vizsgálati eszköztárának.

Abstract

Technologies based on deep neural networks are spreading across every aspect of life. Their advancement in image processing was accelerated by the innovation of GPUs in the 2010s. One of the most important area of application is health care, because solutions that produce great results can make work easier for doctors, and may even save lives. These solutions are mainly present in automatic diagnostic technologies, that could be integrated into the process of medical examination, and are made possible by the growth of databases containing medical images.

In my thesis I classify MRI images of hearts by diagnosis like left ventricular hypertrophy, and if that occurs, than hypertrophic cardiomyopathy (HCM). I apply two types of classification, one for deciding if the patient is normal or ill, and one for deciding if the patient is normal, has HCM, or has some other disease that causes hypertrophy. I attempt fine-tuning some wide-spread pretrained convolutional neural network architectures and feeding the extracted features to recurrent neural nets to make use of the information contained in the structure and movement of the heart.

I run a hyperparameter optimisation on the implemented models and evaluate them on some metrics produced by classification problems, e.g. confusion matrices. In the end I make a comparison of the used convolutional network architectures, and with one of them I show the difference, that some training methods cause in the results.

In the end my best achievements are 90% test accuracy and F_1 score for classification into two classes, and 83% test accuracy and 84% F_1 score for classification into three classes.

In my opinion diagnostic tools based on deep neural networks will be important auxiliary help for doctors in the near future.

1. fejezet

Bevezetés

Ma már szinte az élet minden területén találkozhatunk a gépi tanulás által lehetővé tett technológiákkal. Ezen belül az egyik legnépszerűbb terület a mély neurális hálózatokkal, és az általuk folytatott mélytanulással ("deep learning") foglalkozik. A mély neurális hálók alkalmazási köre napról napra bővül, a modern GPU-k ("Graphical Processing Unit") megjelenése, és architekturális fejlettsége párhuzamos műveletek elvégzése esetén pedig lehetővé teszi szinte bármilyen, a téma iránt érdeklődő személynek, hogy ezekkel kísérletezzen.

Triviális okokból kiemelt fontosságú az egészségügy, mint felhasználási terület, ahol jelenleg leginkább diagnosztikai célokkal vetik be ezt a technológiát. Ezen felhasználás legnagyobb akadálya a megfelelően előkészített, felcímkézett adat alacsony mennyisége. További akadály, hogy ezek az adatok (érthető okokból) a leginkább óvandó személyes adatok közé tartoznak, így a jelentős adatbázisok létrehozásához meg kell teremteni a megfelelő jogi környezetet. Az egészségügyön belül is kiemelkedő irányzat a kardiológia, a szív és az erek betegségeivel foglalkozó tudomány, hiszen egy létfontosságú szervről van szó. Így egy megbízható megoldás a szívbetegségek felismerésének automatizálására valamilyen orvosi diagnosztikai felvételek alapján (MRI, CT, ultrahang) hatalmas segítséget jelentene az ezzel foglalkozó orvosoknak.

Szakdolgozatomban rövid-tengelyi szív MRI (Magnetic Resonance Imaging) felvételek alapján próbálom meg a képeket diagnózisok szerint csoportosítani, először is a balkamrai hipertrófia fennállását, ezen belül főleg HCM (hipertrófiás kardimiopátia) előfordulását detektálni. A rendelkezésre álló felvételeknek része az egészséges páciensek felvételei kontrollcsoport gyanánt, ami egy külön osztályt jelent. Ezt a feladatot előtanított, ismert architekturájú mély neurális hálók felhasználásával kísérlem meg megoldani, mégpedig az MRI felvételek időbeliségét is felhasználva, a páciensek képeit idősorként kezelve, tehát a szív összehúzodó, majd kitáguló mozgását is figyelembe véve.

A dolgozat egy elméleti bevezetővel kezdődik, amiben felvezetem a releváns biológiai ismereteket, továbbá megalapozom a megoldás során használt technologiákat, majd ezek ismeretében bemutatom a szakirodalomban talált eddigi megközelítéseket a témakörben. Ezután a dolgozatban a felhasznált adathalmaz feldolgozásának módját írom le, majd a felhasznált modell felépítését, illetve a tanítás tapasztalatait. Végül a kiértékelés során ismertetem a kapott eredményeket.

2. fejezet

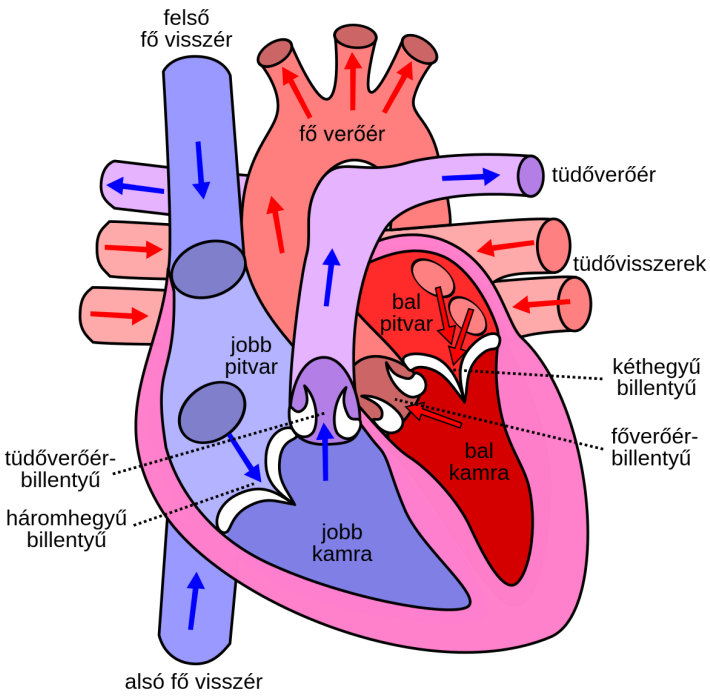
A feladat elméleti háttere

2.1. A feladat orvosi háttere

2.1.1. A szív felépítése

Az emberi test vérkeringésének központi eleme a szív, így ez az egyik legfontosabb szervünk. Lényegében egy szívizomszövetből álló pumpa, ami folyamatosan, ciklikusan összehúzodva, majd ellazulva keringeti a vért a testben.[36]

A szív négy üregre tagolódik, a jobb pitvarra, jobb kamrára, bal pitvarra, és a bal kamrára. A szívciklus során a jobb pitvarba érkezik a nagy vérkörből az oxigénszegény vér, majd a jobb kamrából indul a tüdőbe, a kis vérkörbe. Ezután a tüdőből megérkezik a bal pitvarba, és a bal kamrából kiáramolva ellátja a test szöveteit oxigéngazdag vérrel. Mivel a bal kamra látja el a nagy vérkört, ezért jellemzően ez a szívüreg a legnagyobb térfogatú, illetve a legizmosabb falú. Tehát ezen szívrészek szerepe a szívciklus során létfontosságú.



2.1. ábra. A szív felépítése Forrás:[36]

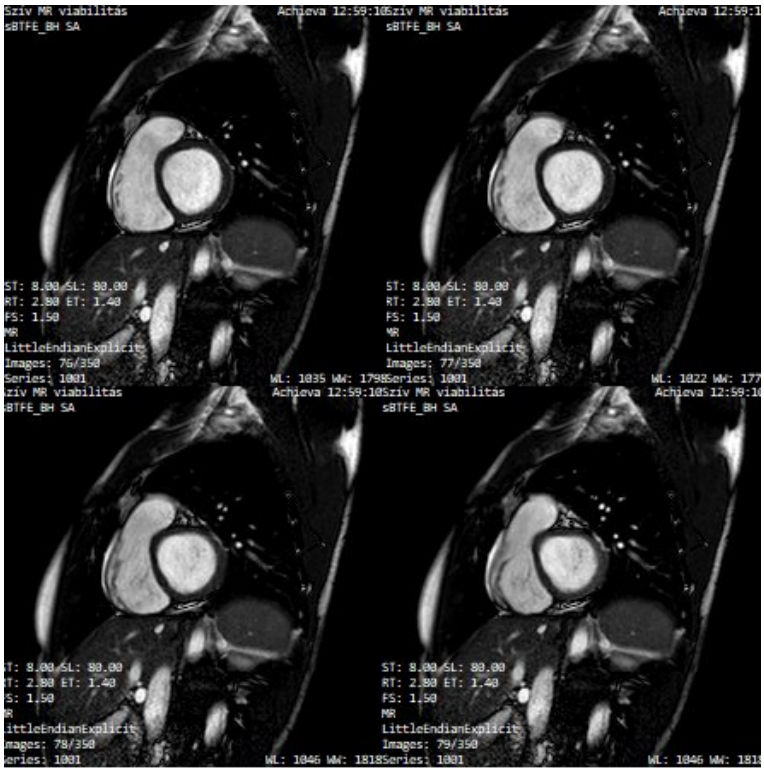
A szívciklus két részre osztható, a "dyastole"-re és a "systole"-re. A "dyastole" során a szívizom ellazul, a pitvarok és kamrák közötti billentyűk kinyílnak, a szívkamrák megtelnek vérrel. A "systole" során az előbbi billentyűk zárnak, majd a verőérbillentyűk kinyilnak a szívkamrák összehúzódásával egyidőben, így a vér kiáramlik a szívből. Bármilyen kóros elváltozás a szívben, ami a szívciklust, illetve a vérkeringést bármilyen formában akadályozza, illetve negatívan befolyásolja, életveszélyes állapotot idézhet elő.

2.1.2. A hipertrófia

A szívkamra hipertófiájáról akkor beszélhetünk, ha a szív valamelyik kamrájának fala megvastagodik, vagyis a szívizom térfogata megnő. A hipertrófia önmagában azt jelenti, hogy a szervezet egy bármely szövetének térfogata megnövekszik, annak ellenére, hogy a szövetet alkotó sejtek száma nem változik, tehát a sejtek egyenkénti térfogata nő meg. Ez a folyamat, amennyiben vázizomzatban történik, még kívánatos is lehet, és a szívizomzatban sem feltétlenül kóros, okozhatja a sportolókra jellemző terhelés, vagy a várandóssággal járó megnövekedett vértérfogat okozta igénybevétel. Azonban, amikor látszólagos ok nélkül válik hipertrófiássá a szívizomzat, akkor egy komoly betegségről, a hipertrófiás kardiomiopátiáról beszélhetünk (HCM). Ez esetben csökken a szívtérfogat, egy szívösszehúzodás során csak kisebb térfogatú vért képes a szív pumpálni, ami hosszabb távon komoly következményekkel járhat, főleg, ha a bal kamra szívizomzatában lép fel hipertrófia, hiszen ennek a kamrának a feladata a teljes test ellátása oxigéndús vérrel. Ennek a betegségnek főleg genetikai okai vannak és öröklödő lehet. A HCM felel a fiataloknál előforduló szívhalál egy jelentős részéért. A szívkamra hipertrófiáját okozhatják egyéb, a HCM-től különböző betegségek is (Fabry-kór, stb).[39]

2.1.3. Magnetic Resonance Imaging (MRI)

Dolgozatom elkészítése során egy MRI felvételekből álló adathalmaz állt rendelkezésemre. Az MRI betűszó a "Magnetic Resonance Imaging", azaz a mágneses magrezonancia képalkotás rövidítése. Az MRI működésének lényege, hogy a pácienst egy erős mágneses térbe téve a testben található hidrogén atomok a mágneses mezőnek megfelelő irányba állnak be. Mivel a test jelentős része vízből áll (65-70 %), szabad, vagy kötött formában, így szinte bármilyen lágy szövet vizsgálható MRI-vel. Ezeket a hidrogén dipólusokat egy gradiens-mező felhasználásával megfelelő, rétegenként különböző frekvenciákkal gerjesztve kitéritik az eddig tartott irányból, majd miközben visszatérnek a mágneses mező által meghatározott irányba, kisugározzák a gerjesztési energiát. Ezt a kisugárzott energiát detektálják, majd rétegenként különböző képeket alkotnak belőlük. Egy kardiológiai, a szívről készült felvétel esetén ezt a képalkotást az időben kiterjesztve kapnak egy sorozatot, így térben és időben is rendelkeznek információval a szív állapotáról. Ez fontos, hiszen nem csak a szív statikus képe, hanem a mozgása is komoly diagnosztikai potenciállal rendelkezik. A szaknyelvben a térbeli vetületeket "slice"-nak, míg az időbelieket "frame"-nek nevezik.[28]



2.2. ábra. Egy rövidtengelyi szív MRI egy slice-a több egymást követő frame-en keresztül

2.2. Technológiai alapok

2.2.1. A gépi tanulás módszerei

Jelenleg három nagyobb gépi tanulási módszerről beszélhetünk. A legelterjedtebb a felügyelt tanulás ("supervised learning"), amit a dolgozatom során én is alkalmazni fogok. Ezen kívül létezik még a felügyelet nélküli ("unsupervised learning"), és a megerősítéses tanulás ("reinforcement learning") is.

A felügyelt tanulás lényege, hogy matematikailag bizonyított, hogy a többrétegű, előrecsatolt (hurokmentes) neurális hálók univerzális approximátorok. [14] Ez azt jelenti, hogy elméletileg bármilyen függvényt képesek megközelítőleg megtanulni, megfelelően nagy szabad paraméterszám (súly) esetén. Tehát, amennyiben rendelkezésünkre áll véges számú bemenet (X) és a hozzájuk tartozó elvárt kimenetek (Y), akkor, amennyiben létezik olyan f függvény, hogy f(X) = Y, akkor ezt a függvényt egy megfelelő háló képes megtanulni, megközelítőleg reprodukálni, azaz ismeretlen bemenetekre is megközelítőleg helyes kimenetet adni. A megfelelő modell megtalálása viszont nem egyszerű feladat. Felügyelt tanítás során egy felcímkézett adathalmaz segítségével vagyunk képesek tanítani a neurális hálót. Ha a kimenet egy folytonos érték, regresszióról, amennyiben egy kategória, akkor osztályozásról, klasszifikációról beszélünk.

A felügyelet nélküli tanulás esetén az adathalmazunk nincsen felcímkézve, ott a háló feladata az adathalmazban mintázatokat felismerni, és ezek alapján különböző

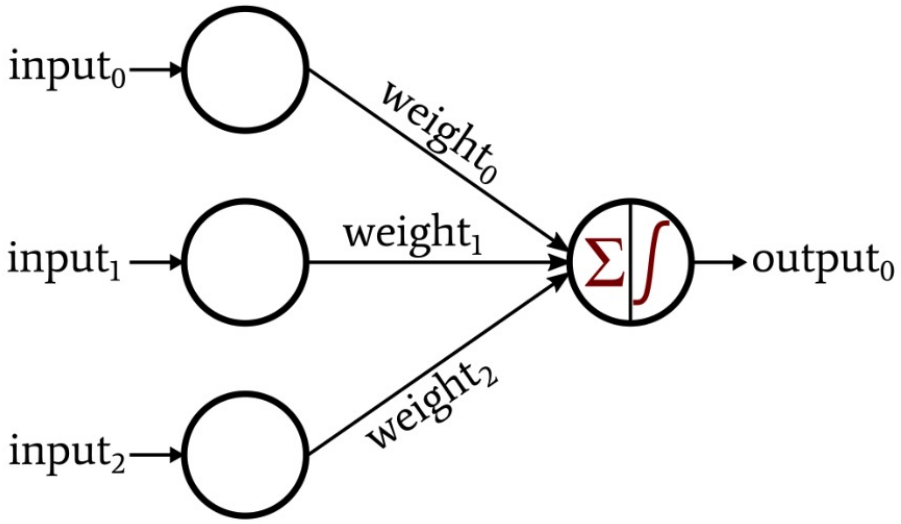
csoportokba rendezni az elemeit. Leggyakoribb módszer a klaszterezés (pl.: k-közép módszer).

Megerősítéses tanulás esetén a neurális háló (ágens), egy cselekvéssorozat végén jutalmat kap annak megfelelően, mennyire volt helyes a cselekvés eredménye. A cél egy olyan stratégia kialakítása, ami maximalizálja a jutalmat.

A három jelentősebb módszer közül a feladat megoldása során a felügyelt tanítással foglalkoztam.

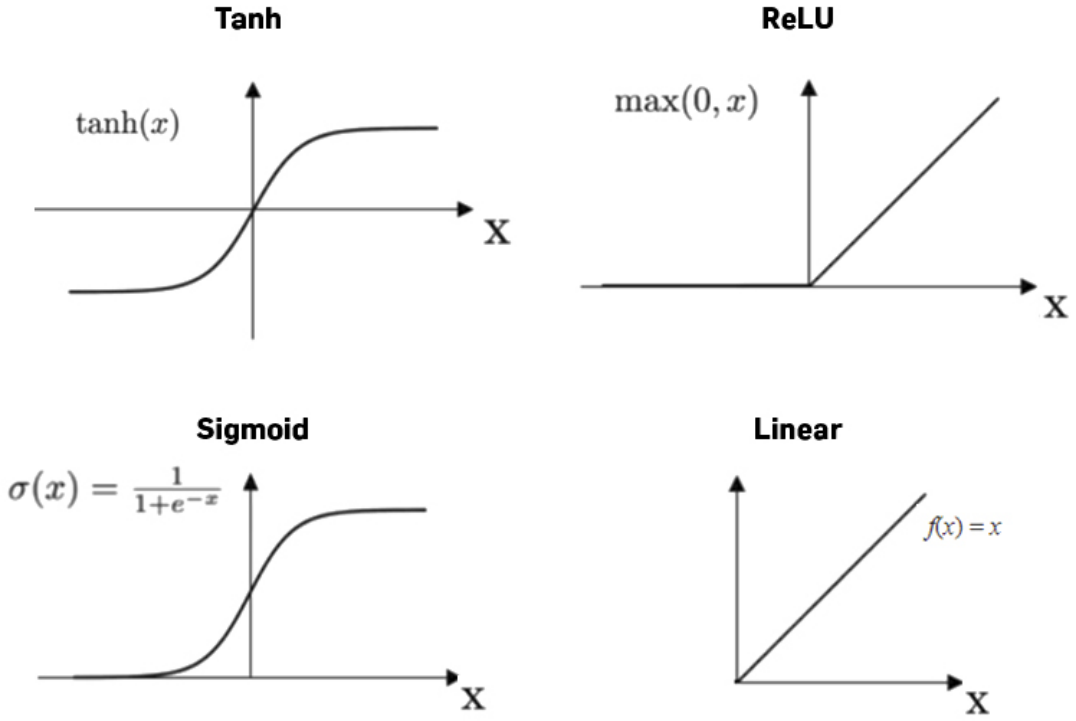
2.2.2. A perceptron és a mesterséges neuron

1958-ban Frank Rosenblatt előállt a perceptron ötletével [32], ami egy, az emberi agyra is jellemző neuron egyszerűsített matematikai modellje. A perceptronba érkezik bizonyos számú bináris bemenet, és ezeket megszorozzuk egy-egy súllyal, majd a kapott értékeket összegezzük. Ehhez hozzáadodik egy "bias", eltolás egy súllyal megszorozva. Amennyiben ez az összeg meghalad egy adott határértéket, akkor a perceptron aktiválódik, "elsül" (egy neuronhoz hasonlóan), tehát a kimenet 1 lesz. Amennyiben a határértéknél kisebb az összeg, akkor a kimenet 0. Ez a konstrukció tanulni is képes, hiszen abban az esetben, ha az elvárt kimenet 1, de a valódi kimenet 0, akkor csak meg kell növelni az olyan bemenetek súlyait, amik 1-esek voltak, ellenkező esetben pedig csökkenteni kell ugyanezen súlyokat. Így a legegyszerűbb logikai függvényeket (AND, OR, NEGATE) képes volt egy perceptron megtanulni, azonban például egy XOR fügvény esetén már több rétegnyi perceptronra lett volna szükség, aminek a tanítása a már említett módszerrel nem lett volna lehetséges, hiszen az csak a kimenet előtti réteg súlyainak változtatására ad iránymutatást.



2.3. ábra. A perceptron felépítése. Forrás: [18]

Az eredeti perceptron továbbfejlesztése a mai neurális hálózatok alapeleme, a mesterséges neuron. Itt a bemenetek már nem csak binárisak lehetnek, illetve nem csupán egy bizonyos határérték felett lesz a kimenet 0-tól különböző, hanem azt egy jellemzően nem-lineáris, deriválható aktivációs függvény határozza meg.

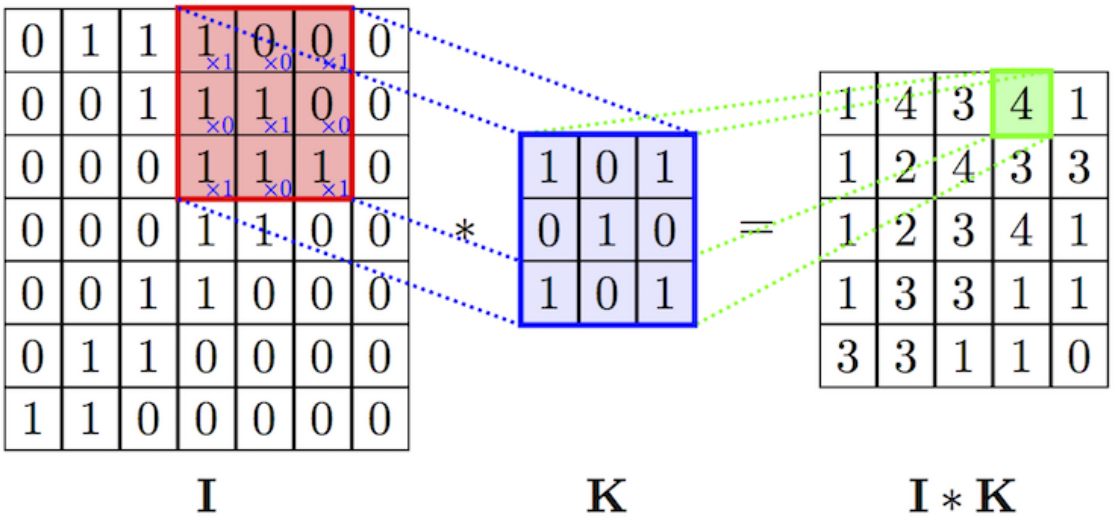


2.4. ábra. A leggyakoribb aktivációs függvények. Forrás: [1]

Ezek az alapelemek már több rétegbe rendezhetőek, hiszen a hibavisszaterjesztésen (backpropagation) alapuló optimalizáló algoritmusok révén taníthatóvá válnak. [34] Egy hibafüggvényt ("loss function"), aminek a paraméterei a valódi és az elvárt kimenet, felhasználva kapunk egy hibát, majd a láncszabályt alkalmazva "visszaterjeszthetjük" ezt a hibát a korábbi rétegekbe, annak megfelelően, hogy melyik súly milyen mértékben befolyásolja a hiba értékét. Az ezt felhasználó, legalapvetőbb tanítóalgoritmus ("optimizer") a gradiens módszer ("Batch Gradient Descent"). Itt a teljes tanító adathalmazon végzett tanítási ciklus ("epoch") után az új súlyok a korábbi súlyok, és a tanulási rátával (egy hiperparaméter) megszorzott, a deriváltakból képzett gradiensvektor különbségével lesznek egyenlőek. Így minimalizálhatjuk a hibafüggvényt, tehát az elvárt és a valódi kimenet egyre közelebb kerül egymáshoz. A gradiens módszer nagy adathalmazok esetén ritkán frissíti a súlyokat, ezért más változata is kialakult. A sztochasztikus gradiens módszer ("Stochastic Gradient Descent", SGD) során minden egyes minta után frissítjük a súlyokat, de ilyenkor lépésről lépésre nagyon megváltozhat a gradiens nagysága és iránya. A középút, a "mini-batch" gradiens módszer ("Mini-batch Gradient Descent") során az adathalmazt adagonként ("mini-batch") adjuk be a modellnek, és az így kapott gradiens alapján frissítjük a modell paramétereit. Az adag mérete ("batch size") a neurális hálózatok egyik jellemző hiperparamétere, azaz a modellt jellemző olyan paraméter, ami nem tartozik a súlyok közé, nem tanítjuk, azonban a tanítás menetére, és a modell teljesítményére nagy hatással lehet.

2.2.3. A konvolúciós hálók

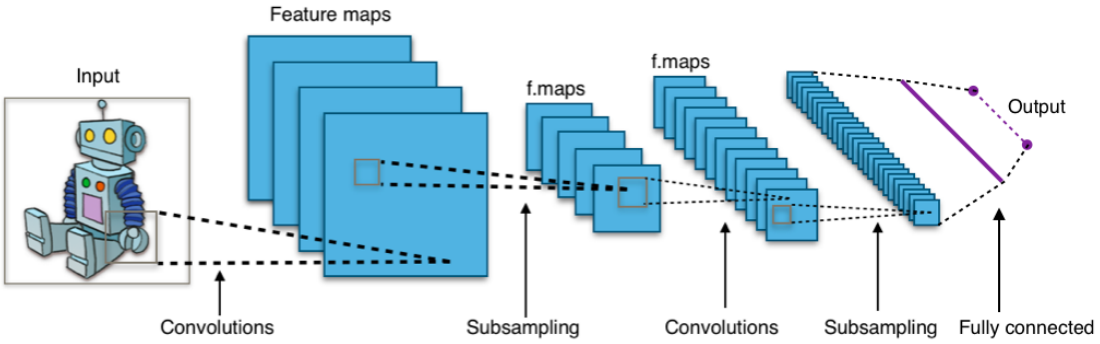
A neurális hálókkal való képfeldolgozás úttörő eseménye az volt, amikor 1989-ben Yann LeCun et al. az Egyesült Államok Postai Szolgálatának (USPS) adathalmazának segítségével, kézzel írott számjegyeket volt képes digitalizálni. [22] Ezt a mérföldkövet konvolúciós rétegekkel sikerült elérni. A konvolúció lényege, hogy egy, a képnél általában jelentősen kisebb méretű (3x3, 5x5, 7x7) "csúszó ablakkal", konvolúciós kernellel (vagy szűrővel) haladunk át a kép pixelein úgy, hogy az ábrán látható módon, "az egymás alá kerülő" pixelek számértékeit szorozzuk össze, majd ezeket szummázzuk, így végül megkapjuk az eredeti kép egy "feature map"-jét, azaz egy olyan mátrixot, ami a kép által tartalmazott információkat egy magasabb absztrakciós szinten tartalmazza. Így, ha ezt 'n' darab szűrővel megtesszük, akkor egy 'n' csatornás "feature map"-et kapunk, amely csatornák különböző jellemzők ("feature"-ök) jelenlétét tartalmazzák.



2.5. ábra. A 2D konvolúció működése. Forrás: [20]

Az előbb említett 'n' a konvolúciós réteg mélysége, aminek egyéb tulajdonságai még az eltolás ("stride"), ami azt adja meg, hogy két konvolúció között hány pixellel toljuk arrébb a szűrőt, illetve a dilatáció, ami kernel két eleme, cellája közti távolságot határozza meg. A réteg kimenetének méretét "zero-padding"-gel, azaz a szélek 0-val való kiegészítésével is módosíthatjuk. A konvolúciós rétegek a lineáris ("fully-connected") rétegnél jóval kevesebb tanítható súllyal rendelkeznek, mert megosztott súlyokkal dolgoznak, hiszen általában a jellemzők (feature) jelenlétét vizsgáljuk, nem a pozíciójukat, tehát csökken a tanítás memóriaigénye.

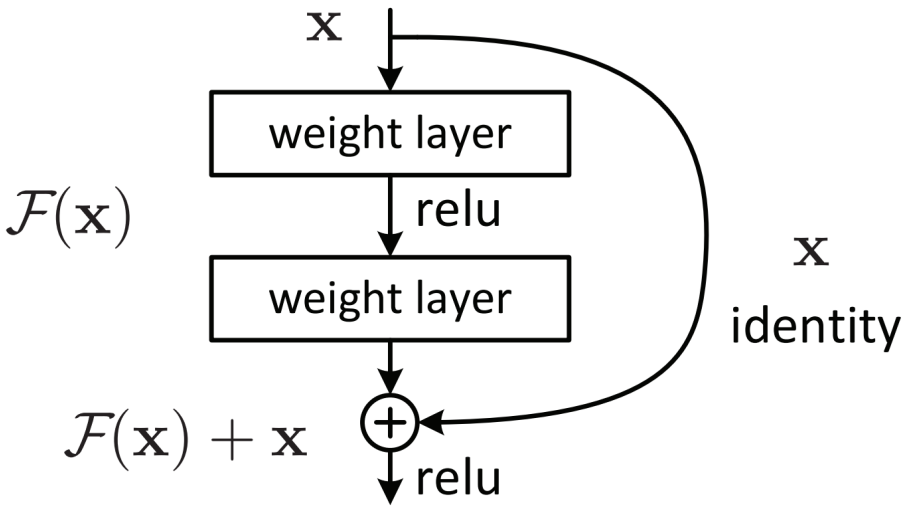
Egy konvolúciós réteg után jellemzően egy úgynevezett "pooling" réteg következik, ami a "downsampling" feladatát látja el. Ennek a lényege az, hogy csökkentsük a kép méretét, ezzel csökkentve a paraméterek számát. Például egy 2x2-es "MaxPool" során egy 2-es eltolású ablakkal haladunk át a képen, és kiválasztjuk a négy szám közül a legnagyobbat. Így felére csökkentjük a kép szélességét és magasságát, a méretét, azaz a paramétereit pedig értelemszerűen a negyedére. Ez hasonlóan működik átlagolással is, illetve bármekkora ablakkal egyaránt.



2.6. ábra. Egy általános konvolúciós háló felépítése. Forrás: [21]

2.2.4. A reziduális hálók

A hálók mélységének növekedésével fellép az eltűnő gradiens ("vanishing gradient") probléma. A hibából számított gradiensvektor egyre kisebb lesz, ahogy visszafele haladunk a kimenettől az első rétegek felé, végül már 0-hoz közelíthet. Így a háló elején álló rétegek súlyai nem változnak, a háló nem tanul. (Hasonló probléma a robbanó gradiens, azonban ekkor a gradiens a végtelenbe tart, ellehetetlenítve a tanítást.) Erre a problémára adnak megoldást a reziduális kapcsolatok. [12] Itt bizonyos rétegek bemenete és kimenete között úgynevezett "shortcut"-okat hozunk létre, azaz a bemenetet hozzáadjuk a következő, vagy valamely később következő réteg kimenetéhez. Így egy mély háló esetén is elkerülhető az eltűnő gradiens probléma.



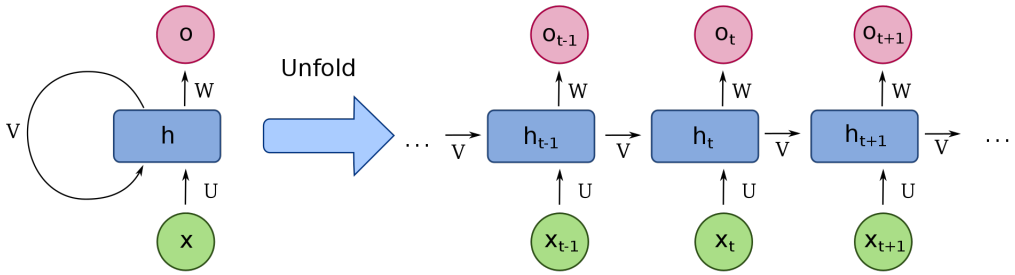
2.7. ábra. Egy reziduális kapcsolat működése. Forrás: [12]

2.2.5. A visszacsatolt hálók

Az eddig ismertetett mély neurális háló architektúrák nem tették lehetővé azt, hogy összefüggő, időrendi kapcsolatban lévő adatsorokat megfelelően feldolgozzanak. Egy adott bemenetre kapunk egy kimenetet, az előtte, illetve utána következő bemenetektől függetlenül. Azonban, ha ez a bemenet kapcsolódik az előző, és a következő

bemenethez, akkor ez plusz információt hordozhat, amit ha képesek vagyunk kezelni, akkor pontosabb kimenetet kaphatunk. Ilyen adatsor lehet például egy szöveg, egy hangsorozat, vagy egy videó.

A visszacsatolt, más néven rekurrens hálók, mint ahogy a nevük is implikálja, hurkokat tartalmaznak, így érve el azt, hogy az előző bemenetek befolyásolják a jelenlegi bemenet feldolgozását. A működés szemléltetésének jellemző módja a háló időben való "kitekerése, kibontása" ("unrolling the network"). Ekkor úgy fogjuk fel a hálót, mintha több példány lenne ugyanabból a hálóból, amiben az előző bemenet absztrakt reprezentációja, és a jelenlegi bemenet egyaránt részt vesz a következő kimenet megalkotásában.



2.8. ábra. A rekurrens hálók "kitekerése". Forrás: [40]

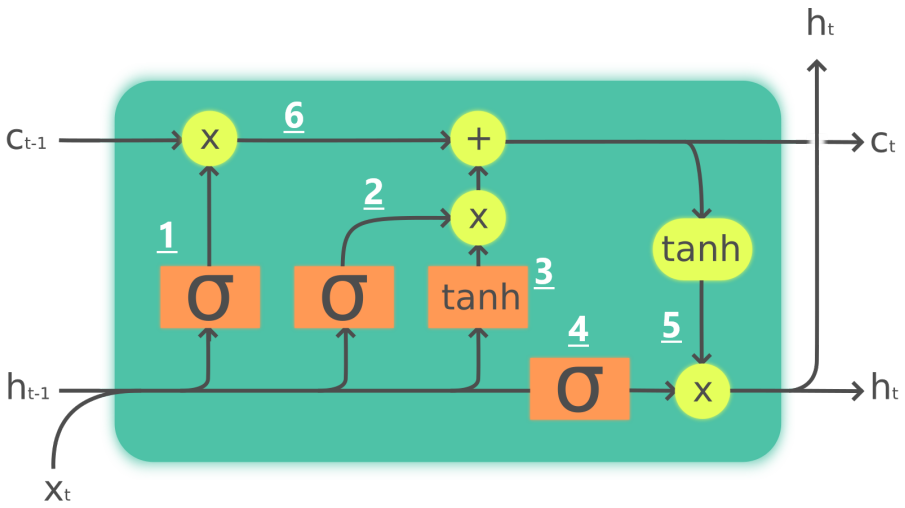
Azonban ennek a megoldásnak is vannak korlátai, nevezetesen, hogy a hosszabb időtartamot felölelő összefüggő adatsorok esetén a pár lépéssel a jelenlegi bemenet előtt levő bemeneteket már nem képesek figyelembe venni. Erre a problémára jelentett megoldást az LSTM ("Long Short Term Memory") [13] bevezetése.

2.2.5.1. Long Short Term Memory

A jellemző, ami lehetővé teszi az információ hosszú időintervallumú tárolását az LSTM-cellában, a cella-állapotot tartalmazó vektor ("cell-state", 6). Ezen kívül a cella felépítésének lényegét a különböző kapuk ("gate"-ek) alkotják. A "felejtő-kapu" ("forget gate", 1), ami a cella-állapot részleges felejtéséért felel, a "bemenet-kapu" ("input gate", 2), ami azt befolyásolja, hogy a cella-állapot mely értékeit, milyen mértékben frissítjük, és a "kimenet-kapu" ("output gate", 4), ami eldönti, hogy az új cella-állapot mely értékeit, és milyen mértékben rakjuk a cella kimenetére. Az ábrán a műveletek Hadamard műveletek [9], azaz elemenként értendőek. A nyilak összefutása vektor konkatenációt, szétfutása másolást jelent. A kapuk szigmoid aktivációjú súlymátrixokból állnak, amelyek kimenetei olyan vektorok, amik elemei 0 és 1 közé esnek.

Az LSTM-cella működése a következő: Az X_t bemenet és az előző cella h_{t-1} kimenete konkatenálva (továbbiakban a cella bemenete vagy bemenet) áthalad a "felejtő-kapun" (1), meghatározva, hogy c_{t-1} előző cella-állapot (6) mely értékeit, milyen mértékben felejti el, illetve tartja meg a cella. A bemenet a "bemenet-kapun" (2) áthaladva befolyásolja, hogy ugyanennek a bemenetnek a hiperbolikus tangens (tanh) függvénye milyen mértékben frissíti c_{t-1}-t, így létrehozva a jelenlegi cella-állapotot c_t-t. Végül a bemenet a "kimenet-kapun" (4) áthaladva meghatározza, hogy

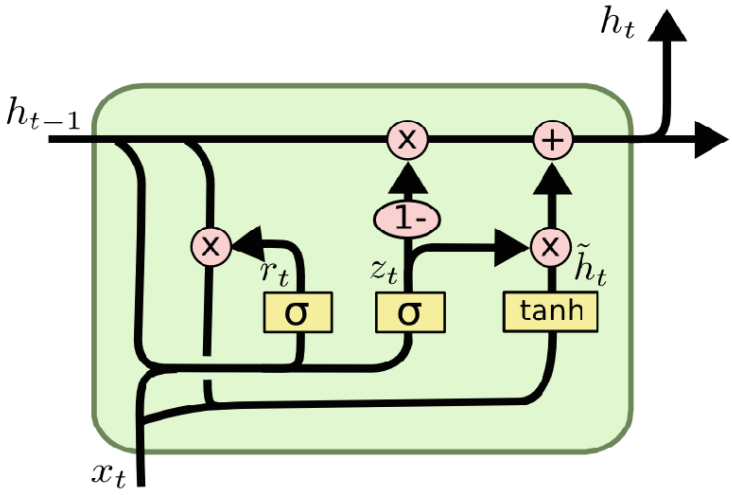
c_t hiperbolikus tangens függvényét (5) milyen mértékben engedjük át a kimenetre (h_t). [26]



2.9. ábra. Az LSTM-cella felépítése. Forrás: [27] kiegészítve a számokkal

2.2.5.2. Gated Recurrent Unit

Az LSTM egy másik, hasonló változata a "Gated Recurrent Unit" (GRU) [5]. A GRU esetén nincsen külön a cella-állapotot tartalmazó vektor. Itt két kapuról beszélhetünk, a "reset-kapuról" ("reset gate", r_t), és a "frissítő-kapuról" ("update gate", z_t), ami az LSTM "bemenet-" és "kimenet-kapujának" "összeolvadása". A "reset-kapu" a felejtést szabályozza, a "frissítő-kapu" kimenete pedig egyszerre azt, hogy az előző cella kimenetének (h_{t-1}), és a jelenlegi bemenet (X_t) konkatenációjának hiperbolikus tanges függvényének, illetve h_{t-1} mely részeinek elemenkénti összege lesz végül a cella kimenete. Ez utóbbi tag h_{t-1} 1-z_t-vel vett Hadamard-szorzatával lesz egyenlő.



2.10. ábra. A GRU-cella felépítése. Forrás: [8]

A GRU-cella működése kevesebb mátrix-művelettel jár, így az LSTM-nél gyorsabb, azonban hasonló eredmények érhetőek el vele, mint az LSTM hálókkal.

2.3. Témaelőzmények a szakirodalomban

A szakirodalomban viszonylag sok publikáció foglalkozik orvosi képek mély neurális hálókkal való elemzésével.

Roth et al [33] 99.8 %-os pontossággal volt képes meghatározni, milyen szervet vagy testrészt (például nyak, tüdő, máj, stb.) ábrázol az adott CT ("Computed Tomography") felvétel, egy konvolúciós neurális háló segítségével. Így megmutatta, hogy a CNN-ek orvosi diagnosztikai felvételek feldolgozására is alkalmasak, annak ellenére, hogy az egy osztályba tartozó képek között is jelentős különbségek figyelhetőek meg. Ezen a képek megfelelő augmentációjával segített, például forgatással és egyéb véletlenszerű transzformációkkal.

Shahzadi et al [35] agyi MRI felvételeken kíséreltek meg agydaganatokat (glioma) klasszifikálni, pontosabban ezeket előrehaladottság szerint osztályozni. Mindezt úgy, hogy az MRI képek "slice"-aiból egy előtanított VGG architektúrájú konvolúciós hálóval kinyerték a jellemzőket, majd egy rekurrens hálóba ezt továbbítva, kihasználták az MRI felvételek három dimenziós voltát, azaz a "slice"-ok egymáshoz képesti elhelyezkedését is felhasználta a diagnózisnál. Viszonylag csekély darabszámú (60) adathalmaz felhasználásával is 84 %-os pontosságot értek el.

Poudel et al [29] ugyanezt a megközelítést alkalmazta kardiológiai MRI felvételekre, továbbra is a térbeliséget felhasználva, mint rekurrens háló bemenet, hogy a szív bal kamráját szegmentálja, azaz jelen esetben a bal kamra-kontúrt berajzolja.

Zhang et al [41] már a szív mozgását is kihasználta a szegmentálás során, hiszen az MRI felvételek idősorosan is értelmezhetőek, mindezt sertések szívéről alkotott képekről. Ebben a hálóban konvolúciós háló gyanánt egy előtanított ResNetet használtak, a rekurrens rétegeket pedig ConvLSTM-cellák alkották, amiknek az a különlegessége, hogy a szokványos LSTM-cellára jellemző mátrixszorzások helyett konvolúció történik.

Bernard et al [4] 2018-ban végigvette az eddig elért eredményeket a szívről készült MR felvételek neurális hálókkal való elemzésének területén. Arra a következtetésre jutott, hogy az eredmények azt vetítik előre, hogy a közeljövőben, amennyiben nem csak a diagnosztizált betegség (90+ %-os pontosság), hanem a diagnózis magyarázata is megfelelő pontossággal lesz egy háló kimenete a megfelelő szívkontúrokkal, továbbá számszerű adatokkal (például szívtérfogat) együtt, akkor akár egy automatikus MRI analizáló megoldás könnyen integrálható lenne az MRI felvételt produkáló folyamatba.

3. fejezet

A fejlesztési környezet

A feladat során a Facebook AI Research által létrehozott PyTorch 1.7.0.[1] keretrendszert használtam a modell tanításához (ami az eredetileg Lua szkriptnyelv alapú Torch Python implementációja), ezen belül főleg a torchvision állományt, amit képekkel való munkára készítettek, így tartalmaz több népszerű, előtanított konvolúciós hálót, illetve képtranszformációt, augmentációs függvényt. Mindezt a Google Colab[2] felhasználásával tettem, amit Google Drive-on keresztül összekötöttem a lokális Visual Studio Code fejlesztőkörnyezettel. Értelemszerűen a feladat során használt programozási nyelv a Python volt. Lett volna lehetőség az AUT tanszék egy szerverét is tanításra használni, azonban szükség lett volna előre regisztrálni a szándékot. Továbbá a Colab-on elérhető GPU-k jellemzően több memóriával rendelkeznek (16 Gb), mint a tanszéki szerveren elérhető Nvidia GTX 1070-es videókártya (8 Gb). Előfordult, hogy 16-nál nagyobb batch-méret esetén, 10 gigabyte-nál nagyobb memóriafelhasználás mellett elszállt a tanítás. A Colab esetén is vannak felhasználási limitációk, amik nincsenek pontosan definiálva, ezért többször volt olyan alkalom, amikor pár napig nem tudtam GPU-val ellátott környezethez kapcsolódni, mert a napi többszöri kísérlet mellett valószínűleg átléptem valamilyen limitet. Továbbá viszonylag hamar, fél óra után a kapcsolat a Colab szerverrel automatikusan megszakad, így ennél az időtartamnál hosszabb folyamatok esetén, például hiperparaméter optimalizáció során, egy apró JavaScript szkriptre volt szükségem, ami percenként belekattintott a webes felületbe, így megakadályozva a "timeout"-ot. Ezeknél a kisebb-nagyobb kellemetlenségeknél nagyobb problémába nem ütköztem, ami a fejlesztési környezettel, vagy az infrastruktúrával lett volna kapcsolatos.

[1] A PyTorch hivatalos oldala: `https://pytorch.org/`

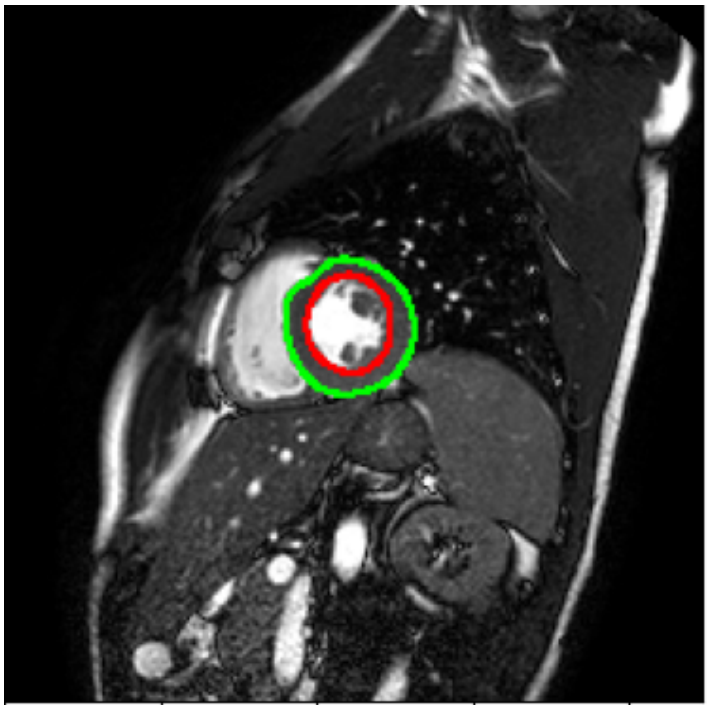
[2] A Google Colab hivatalos oldala: `https://colab.research.google.com/`

4. fejezet

Az adatok feldolgozása

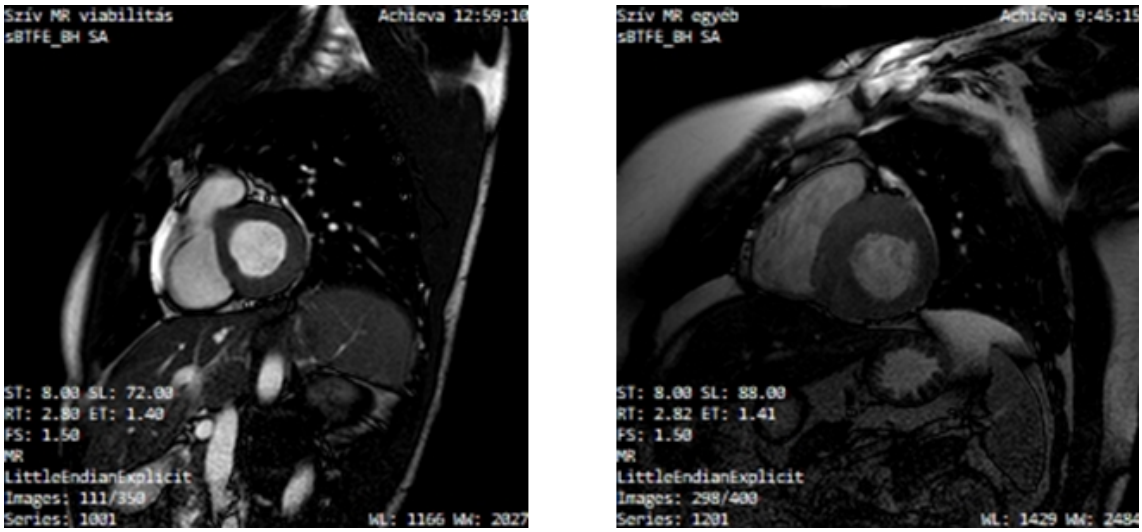
4.1. Az adathalmaz bemutatása

Az adathalmaz a Városmajori Szív- és Érgyógyászati Klinika jóvoltából volt elérhető számomra, ami 699 anonimizált páciens rövidtengelyi (short-axis) szív MR képeit tartalmazza, illetve az ezekhez tartozó, orvosok által berajzolt szívkamra kontúrjait. Mivel a dolgozat során balkamrai képekkel foglalkoztam, ezért a kontúrok közül csak a bal kamra kontúrjait használtam fel. Ezek az epikardiumot, azaz a szívkamra legkülső rétegét, illetve az endokardiuomat, azaz a szívkamra falának belső rétegét határolják. Az epikardium kontúrja jellemzően a szívciklus "dyastole" fázisában van berajzolva, azaz akkor, amikor a szív kitágul, a bal szívkamra térfogata a legnagyobb, az endokardium kontúrja pedig mind a "dyastole", mind a "systole" fázisban adott, tehát akkor is, amikor a szív összehúzodik, és a bal szívkamra térfogata a legkisebb.



4.1. ábra. Egy egészséges szív MRI képe berajzolt balkamrai kontúrokkal. A zöld kontúr az epikardiumot, a piros az endokardiumot jelöli.

A felvételek egy jelentős része sportolóktól, illetve gyerekektől származik, akiket a feladat megoldásához nem célszerű felhasználni, hiszen bár hipertrófiára hasonlító képeket produkálnak, azonban ezekben az esetekben a bal szívkamra falának megvastagodása nem kóros eredetű. Ezeket a mintákat eldobva körülbelül 400 darab páciens képei maradtak meg, amiket felhasználhattam, ami elég kicsi adathalmaznak tűnt kép osztályozás szempontjából. Azonban az általam olvasott cikkek adathalmazainak pácienssz̈ámai körülbelül 200-220 darabnál maximalizáltak, tehát a számomra elérhető adathalmaz viszonylag nagynak számít. A képek szürkeárnyalatosak, és túlnyomó többségben 224x224 pixelből állnak, azonban vannak olyanok, amik bár 1:1 arányúak, de valamivel kisebbek, vagy nagyobbak.

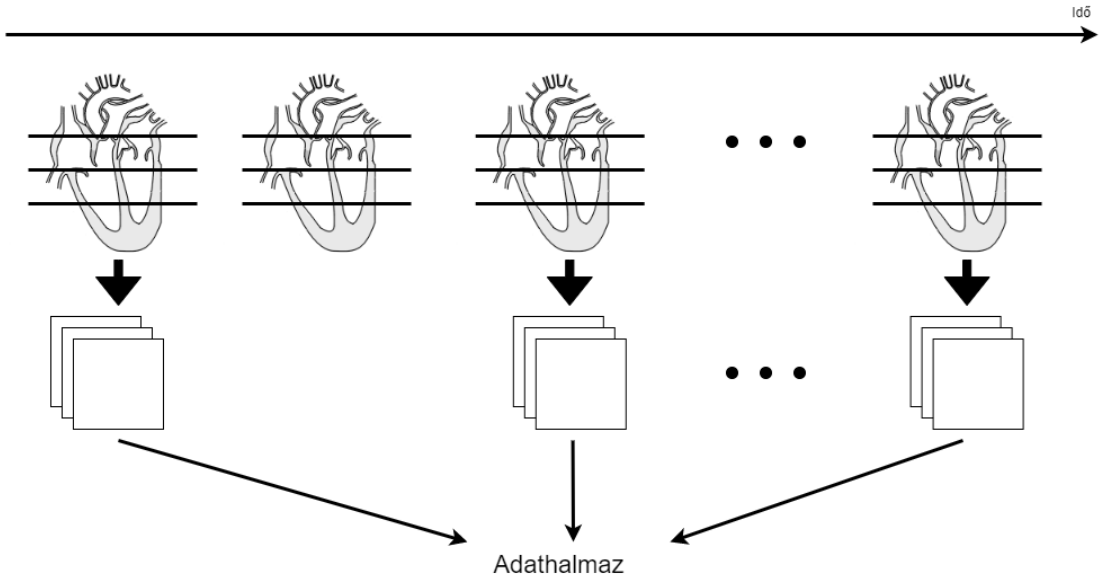


4.2. ábra. Példa MRI az adathalmazból. Bal oldalt egy egészséges szív, jobb oldalon egy hipertrófiás (HCM-es) szív látható.

4.2. Előfeldolgozás

Az MRI képek "slice"-okból, és "frame"-ekből állnak, általában 13-16 "slice"-ból, és 25 "frame"-ből. A rövidtengelyi szív MRI képeinek "slice"-ai a rövidtengellyel párhuzamosan, a hosszútengelyre merőlegesen készült metszetek. A "frame"-ek pedig az MRI felvétel időbeli függvényének leképezései. Azaz egy "frame"-ben található "slice"-ok meghatározzák a szív állapotát egy adott időpontban, a következő "frame" "slice"-ai pedig a következő időpontban. Mivel én a megoldásom során mind a szív térbeli, azaz "slice"-ok által meghatározott, és az időbeli, azaz a "frame"-ek által meghatározott változásait is szerettem volna felhasználni az információtartalmuk végett, ezért egy páciens felvételei közül minden második "frame"-et, és azon belül a szív felső, középső, és alsó harmadából is egy-egy "slice"-ot választottam ki egy adatfeldolgozó szkript megírásával. Így egy pácienshez 13 darab "frame", és mindegyik "frame"-hez három darab "slice" tartozik, amiket egy közös "numpy array"-ben összefűztem ("stack"), mert a CNN architektúrák, amiket felhasználtam, mind háromcsatornás képeket várnak a bemenetükre, míg az MRI képei szürkeárnyalatosak, azaz egycsatornásak. Így 13 hosszúságú idősorokat kaptam, amiket a szkript pá-

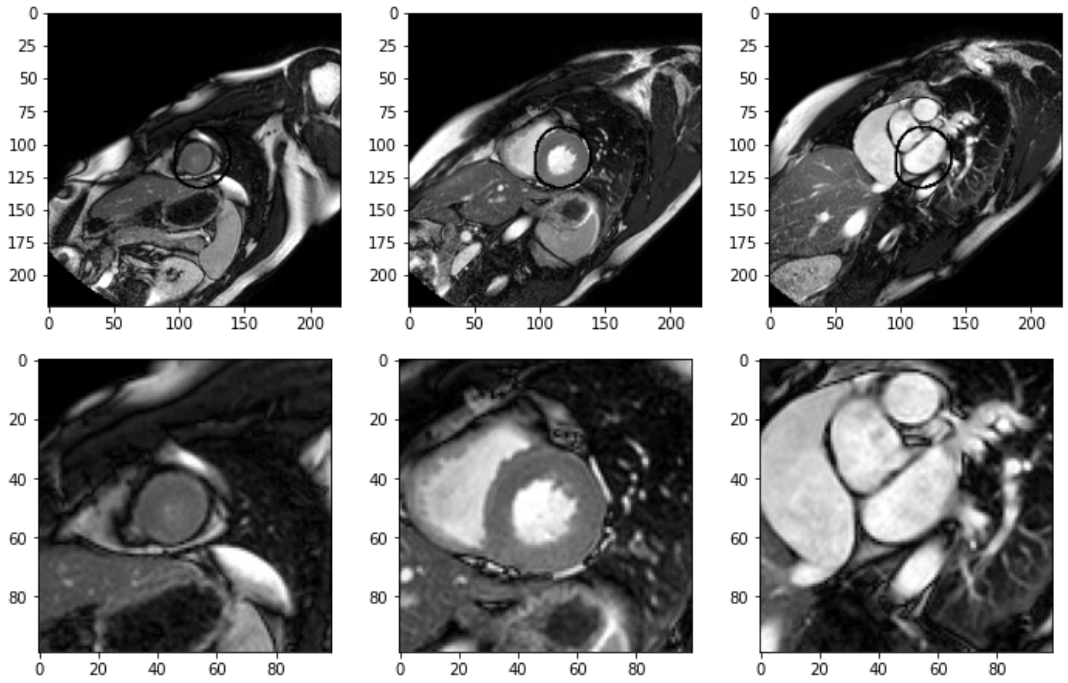
ciensenként a "pickle"[1] (a Python nyelv objektumszerializációs függvénykönyvtára) segítségével fájlokba szerializált.



4.3. ábra. Az MRI felvételekből egy páciens adathalmazának előállása egy sematikus ábrán.

Ezen kívül megpróbálkoztam azzal is, hogy nem a "nyers", adott méretű MRI képeket vettem a tanításhoz, hanem belevágtam ezekbe, hiszen ahogy a 4.1-es ábrán is látszik, az esetek nagy többségében a képeknek csak egy kis részét teszi ki a vizsgált bal szívkamra, így a kép többi része zajként funkcionál, csak nehezíti a tanulást. Ennek megfelelően vettem az összes rövid tengelyi bal szívkamrát jellemző kontúrokat, endokardiumot és epikardiumot egyaránt minden pácienshez. Az epikardiumot jelölő kontúr természetéből fakadóan nagyobb, mint az endokardiumot jelölő, ezért ha az adott páciens esetén rendelkezésre állt, akkor azt használtam, egyébként pedig a kisebb endokardiumot. Majd ezeken végigiterálva megkerestem a páciens összes kontúrjának összes pontjának legnagyobb, illetve legkisebb x és y koordinátájit. Az ezekből kapott határoló téglalapot egy, a téglalap hosszabb oldalával megegyező oldalú négyzetté bővítettem. Majd attól függően, hogy a kontúr epi- vagy endokardiális volt-e, egy bizonyos konstanssal megszoroztam az oldalait, hogy a feladat szempontjából fontos bal szívkamra mindenképpen rajta legyen a kivágott képen. Ennél a lépésnél sajnos elkövettem egy hibát, mégpedig azt, hogy a legkisebb koordinátákat is pozitív számmal szoroztam meg, így a kivágott négyzetet eltoltam, ezzel sok olyan képet is létrehozva, amin rajta sem volt a szív. Szerencsére, amikor kirajzoltam a kivágott képeket, akkor észleltem a hibát és sikerült kijavítani a hibás kódrészeket. Mivel az MRI felvétel készítése során a páciens nem mozdul, és a szívciklus során sem történik jelentős elmozdulás, ezért feltehető, hogy a kivágott téglalap minden "frame" és "slice" esetében tartalmazni fogja a vizsgálandó bal szívkamrát.

[1] A pickle könyvtár hivatalos dokumentációja: `https://docs.python.org/3/library/pickle.html`



4.4. ábra. Egy páciens egy "frame"-énekhárom "slice"-a a bal kamra legnagyobb kontúrjával, és a belőlük kivágott képek.

4.3. Az adatok csoportosítása

HCM	Normal	U18-m	adult-m-sport	adult-f-sport
188	179	109	100	49

U18-f	Amyloidosis	Fabry	Aortastenosis	EMF
29	25	7	7	6

4.1. táblázat. Az eredeti adathalmaz címkéi, és azok számossága (m: male, f: female, U18: under 18).

Az eredeti adathalmaz tíz különböző címke alapján volt csoportosítva. Ezek közül, mint azt az egyik előző bekezdésben már említettem, a sportolókhoz és a gyermekekhez tartozó felvételeket nem használtam fel. Később észrevettem, hogy néhány páciens esetében, amikor minden második "frame"-et választottam ki, 25 "frame" került az adathalmazba, mivel az ő esetükben eredetileg 50 "frame" volt. Ezek fele teljesen fekete, feltehetően valami hiba történt az MRI felvétel elkészítése során, ezért az eljárást megismételték. Ez a probléma 5 pácienst érintett, ezeknek a sötét "frame"-jeit eldobtam, így végül náluk is 13 maradt. Ezeken kívül találtam még egy pácienst, akinek csak fekete képei voltak, így őt is kénytelen voltam kihagyni az adathalmazból. Így végül egy 406 darab páciensből álló adathalmazzal dolgoztam.

HCM	Normal	Amyloidosis	Aorta-stenosis	Fabry	EMF
184	178	24	7	7	6

4.2. táblázat. A végül felhasznált adathalmaz címkéi, és azok számossága.

Mivel a feladatom főleg az volt, hogy egyáltalán a hipertrófia jelenlétének tényét megállapítsam, esetleg ezen belül a hipertrófiás kardiomiopátia fennállását, ezért kétféle osztályozást vezettem be a feladat megoldása során. Az első esetben két osztályba soroltam a pácienseket. "Normal"-ba az egészségeseket, "Other"-be pedig azokat, akiken megfigyelhető a hipertrófia bármilyen formája.

Normal	Other
178	228

4.3. táblázat. A csoportok számossága kettő csoport esetén.

A második osztályozás során az előbbi kettő osztályt kiegészítettem egy harmadikkal, egy "HCM" címkéjűvel. Ide kerültek át az "Other" csoportból azok, akiknek a címkéjük "HCM" volt.

Normal	HCM	Other
178	184	44

4.4. táblázat. A csoportok számossága három csoport esetén.

Látható, hogy mindkét osztályozási módszer esetén kiegyensúlyozatlan a csoportok elemszáma, azaz az elemszámok aránya az első esetben nem 50 : 50, illetve a második esetben nem 33 : 33 : 33. Ez problémákat okozhat a mély neurális háló tanítása során, hiszen így a háló könnyen elfogult ("biased") lehet azon osztály irányába, amiből több minta található a tanítóadathalmazban, hiszen elég jó pontosságot érhet el azzal, hogy minden elemre a többségi osztályt prediktálja. Két osztály esetén ez a kiegyensúlyozatlanság nem olyan jelentős, azonban a három osztály esetében a kisebbségi csoport elemszáma nagyjából a negyede a másik kettőnek.

4.4. Tanító, validációs és teszt adathalmazok

Az adathalmazt a gépi tanulást alkalmazó megoldásoknál általános módon tanító, validációs, illetve teszt diszjunkt adathalmazokra bontottam, az elterjedt 80%-10%-10% arányban. Törekedtem arra, hogy az eredeti adathalmaz osztályarányai jellemezzék mindegyiket, azaz az egy osztályhoz tartozó minták is 80%-10%-10% kerültek szétosztásra, de ezek sorrendje meg lett keverve. Ezt a szétbontást csak egyszer csináltam meg, majd az adathalmazokat különböző fájlokba mentettem ki, ugyanis viszonylag sok időbe telt, amíg az egyes páciensek felvételeit tartalmazó "pickle" fájlokból egyenként beolvastam a képeket. A tanító, validációs és teszt adathalmazokat tartalmazó fájlok betöltése már nem tartott jelentősebb ideig.

A tanító adathalmaz tartalmazza azokat a mintákat, amiken a mély neurális háló tanul, azaz amikre kiszámolja a hibát, és ezt a hibavisszaterjesztő algoritmus során visszaterjeszti a korábbi rétegekbe. Minden tanítási ciklus (epoch) után a modellt kiértékeltem a validációs adathalmazon, ekkor a modell nem terjeszti vissza a kiszámolt hibát, csak prediktál a bemeneti mintákra. Az ezekből kapott metrikák alapján lehet optimalizálni a neurális háló hiperparamétereit (például tanulási ráta, satöbbi). Majd, a tanítás végeztével, amikor a validációs metrikákkal meg vagyok elégedve, megtörténik a kiértékelés a teszt adathalmazon. Erre azért van szükség, mert a hiperparaméterek finomhangolása során lehetséges, hogy a validációs adathalmazon is túltanul a háló közvetve, és az adott konfiguráció csak erre a szűk adathalmazra ad jó eredményt. Az teszt adatokon kapott metrikák használhatóak a különböző modellek teljesítményei közötti összehasonlításra. Ezekből láthatjuk, hogy az adott háló mennyire képes generalizálni, általánosítani, azaz a tanult minták alapján ismeretlen mintákra is helyes predikciót adni.

4.5. Adat augmentáció, adat dúsítás

A limitált darabszámú páciensből álló adathalmaz véleményem szerint dúsításra szorult, azaz mesterséges képek generálását tartottam szükségesnek, hogy a túltanulást minél sikeresebben el tudjam kerülni. Ehhez a PyTorch torchvision nevű "package"-ében található transforms[2] állományt használtam fel. Először az eddig "numpy array"-ként tárolt képeket "PIL Image"-gé[3], majd az eddig különböző méretű képeket azonos méretűvé alakítottam, véletlenszerűen elforgattam, maximálisan egy bizonyos megadott szöggel, illetve random módon egy nem sokkal kisebb méretű képet vágtam ki belőlük. Végül a képeket "torch.Tensor"-rá alakítottam, majd normalizáltam az egész adathalmazra jellemző átlaggal, és átlagos szórással. Ezeket a műveleteket minden mintavételezésnél elvégzi a keretrendszer, így minden tanítási ciklusban egy kicsit különböző tanító adatok fognak a modell rendelkezésére állni.

4.5.1. Az adathalmaz kiegyensúlyozása

Egy kiegyensúlyozatlan adathalmaz kiegyensúlyozására több módszer is létezik. Ezek közül én az újramintavételezést ("resampling"), azon belül is a túlmintavételezést ("oversampling"), illetve ezen kívül a költségfüggvény alapú kiegyensúlyozást próbáltam ki.

Az újramintavételezés során az eredeti adathalmaz osztályarányait a mintavételezés módosításával próbáljuk meg ellensúlyozni. Ennek egyik módszerre az alulmintavételezés ("undersampling"), aminek a lényege az, hogy abból az osztályból, amiből több minta áll rendelkezésre, nem használunk fel minden mintát, így az osztályok aránya egyenlő lehet. Azonban, szerintem ez a rendelkezésre álló adathalmaz szűkössége miatt, ebben az esetben nem lett volna célszerű. Ezért az újramintavételezés másik módszerét próbáltam ki, a túlmintavételezést. Itt a kisebbségben lévő osztály mintáit egy tanítási ciklus során többször is beadjuk a hálónak. Mivel az augmentáció minden mintavételezés során véletlenszerű módon megtörténik, ezért

[2] A torchvision.transforms hivatalos oldala: `https://pytorch.org/docs/stable/torchvision/transforms.html`

[3] A pillow (PIL) hivatalos oldala: `https://pillow.readthedocs.io/en/stable/`

ezek a többször vett képek várhatóan különbözőek lesznek, így az eredmény hasonló lehet, mintha valóban más mintákat használnánk. [11][10]

A túlmintavételezést a "torch.utils.data.WeightedRandomSampler" osztállyal oldottam meg, ami a minták osztályainak előfordulási valószínűségének reciprokát rendeli az egyes mintákhoz, így elérve, hogy a kisebbségi osztályba tartozó minták nagyobb valószínűséggel kerüljenek be a soron következő "mini-batch"-be.

A költségfüggvény alapú kiegyensúlyozást a költségfüggvények bemutatásakor, a 7.4-es fejezetben részletezem.

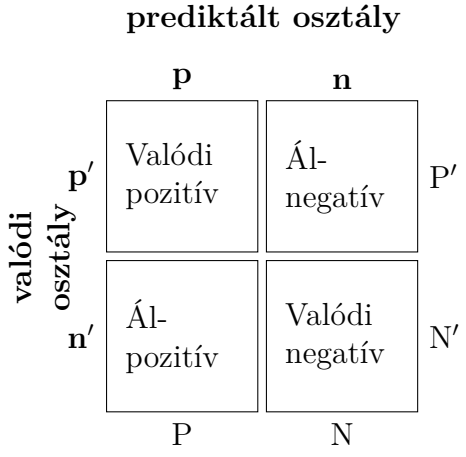
5. fejezet

Kiértékelés

5.1. Kiértékelési metrikák

5.1.1. Pontosság

A pontosság ("accuracy") a neurális hálók esetén jellemzően a legalapvetőbb kiértékelési metrika. Értéke egy osztályozó modell esetében azt adja meg, hogy az adott modell a kapott minták mekkora hányadát osztályozta helyesen, azaz a helyesen osztályozott minták száma elosztva az összes minta számával. Azonban ez a metrika egy kiegyensúlyozatlan adathalmaz esetén, mint például az, amivel én dolgoztam, igencsak félrevezető lehet. Például egy olyan esetben, ahol az adathalmazban két osztályunk van és a minták darabszámainak aránya 1:99, akkor amennyiben a modell csak a számosabb osztályt prediktálja, akkor is 99%-os pontosságot produkál.



5.1. ábra. Példa egy bináris tévesztési mátrixra.

5.1.2. Tévesztési mátrix

A tévesztési mátrix ("confusion matrix") egy megfelelő vizualizáció kiegyensúlyozatlan adathalmazon tanított modellek kiértékelése esetén. Ez egy mátrixban összeveti, hogy melyik osztályba hány mintát prediktált a modell, illetve, hogy mik a minták valódi címkéi ("ground truth"). Így osztályonként megkapjuk a valódi pozitívok számát ("true positives", TP), azaz a helyesen az adott osztályba prediktált mintákat,

az álpozitívok számát ("false positives", FP), azaz az osztályba prediktált, de valójában nem az osztályba tartozó mintákat, továbbá hasonlóan a negatívokat is. A valódi negatívok ("true negatives", TN) azok a minták, amiket a modell nem az osztályba prediktál, és valóban nem is tartoznak bele, az álnegatívok ("false negatives", FN) pedig, amiket nem az osztályba tartozónak prediktál, de valójában beletartozik. A feladat megoldása során vizsgáltam a modellek által produkát tévesztési mátrixokat, mert ezekből látható, hogy melyik osztályt, mennyire ismeri fel a háló.

A feladat megoldása során a metrikák kiszámítására a scikit-learn[1] Python könyvtárat használtam. Ezen belül a tévesztési mátrixokat a "sklearn.metrics.confusion_matrix" függvény felhasználásával kaptam meg.

5.1.3. Felidézés és a precizitás

A felidézés ("recall"), illetve a precizitás ("precision") a TP, FP, TN, FN értékeket felhasználva ad szemléletes értékelést a modell teljesítményéről. A felidézés azt adja meg, hogy egy adott osztályba tartozó minták mekkora részét sorolta a modell az adott osztályba. Tehát, mint a feladatom esetében, ha egy betegséget szeretnénk diagnosztizálni, akkor fontos, hogy a betegség osztályába tartozó minták minél nagyobb hányadát helyesen klasszifikáljuk, azaz a modell felidézése legyen minél nagyobb.

$$Felidézés = \frac{TP}{TP + FN}$$

A precizitás azt adja meg, hogy egy adott osztályba tartozónak prediktált elemek mekkora része tartozik valójában az osztályba.

$$Precizitás = \frac{TP}{TP + FP}$$

5.1.4. Az F_1 érték

Az F_1 érték a felidézés és a precizitás harmonikus közepe.

$$F_1 = 2 * \frac{Precizitás * Felidézés}{Precizitás + Felidézés}$$

Az F_1 érték maximalizálása osztályozás feladatok esetén egy jó cél, mert célszerű a felidézés és a precizitás közötti egyensúlyt megtalálni, hiszen az egy optimális megoldás, ami a lehető legtöbb hipertrófiás illetve HCM-es beteget felismer úgy, hogy a betegnek prediktált minták között minél nagyobb legyen a valóban beteg páciensek száma. Ezért ezt a metrikát választottam a fő értékelési szempontnak a lehetséges modellek összehasonlítása során. A feladat megoldásában ennek számítására a "sklearn.metrics.precision_recall_fscore_support" függvényt használtam, ami a pecizitást, és a felidézést egyaránt visszaadja. Ezt a függvényt a "weighted" paraméterrel alkalmaztam, mert ez az osztályok közötti elemszámkülönbségeket is figyelembe veszi az értékek átlagolásakor.

[1] A scikit-learn hivatalos oldala: `https://scikit-learn.org/stable/index.html`

6. fejezet

A használt modellek bemutatása

6.1. Transfer learning

A szakdolgozat feladat megoldása során előtanított konvolúciós hálók felhasználásával kísérleteztem. Próbálkoztam egy saját, egyszerű, pár réteges konvolúciós architektúrával is, azonban az nem produkált értékelhető eredményt. Ezért próbáltam már más képklasszifikációs feladatok esetén bevált, széleskörűen használt architektúrájú neurális hálókat találni, majd ezeket a saját adathalmazom és feladatom által állított keretekre igazítani. Ezt az eljárást, az előtanított hálók saját igényekre szabását a szakirodalomban "transfer learning"-nek nevezik.

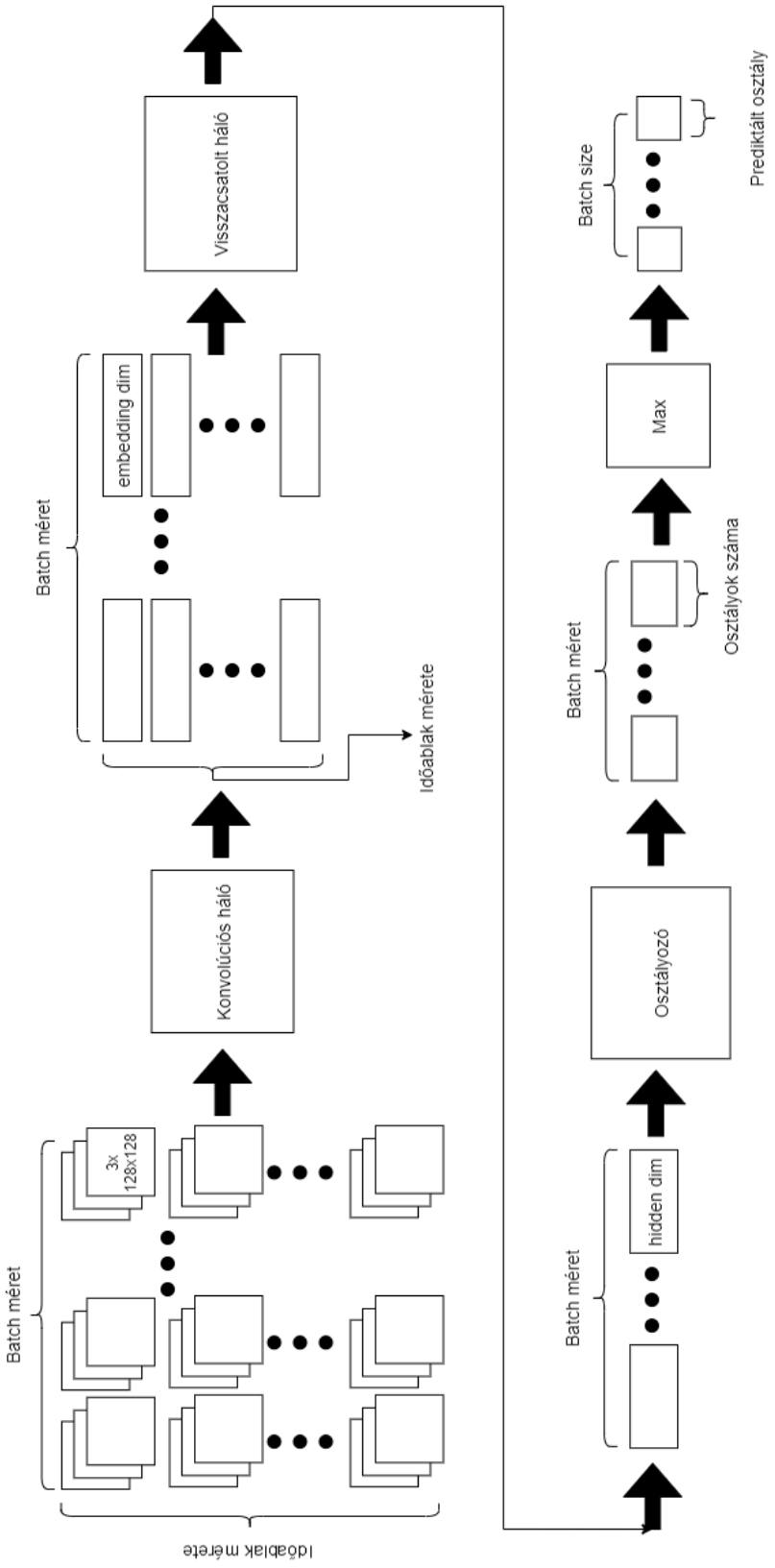
A "transfer learning" legnagyobb előnye, hogy jellemzően képes alacsony mintaszámú tanítóadatbázisok esetén is jó eredményt elérni. Ez annak tudható be, hogy az alapvető "feature"-ök, jellemzők (élek, esetleg formák) a képek jelentős részén hasonlóak, a mély neurális háló kezdeti rétegei ilyen általános jellemzőket tanulnak meg, míg a vége felé a tanult "feature"-ök egyre specifikusabbak lesznek. A modell tanítása során egyszerűbb a súlyokat finomhangolni, mintha random generált számokkal inicializálnánk a modellt. Az általam használt előtanított neurális hálók, az ImageNet[1] nevezetű képadatbázison lettek előtanítva, ami több mint 12 millió, jellemzően természetes képből áll, körülbelül 1000 különböző osztályba csoportosítva. Felmerül a kérdés, hogy az orvosi és MRI képek klasszifikációjában mennyire segíthet a természetes képeken előtanított háló, hiszen ezek a képek ránézésre nagyon különböznek az ImageNet képeitől. Azonban Tajbakhsh et al [37] arra jutott, hogy ilyen esetben is hamarabb konvergál az előtanított modell tanítás során, mint random inicializált súlyok esetén.

Két megközelítése létezik a "transfer learning"-nek. Az egyik esetben az előtanított konvolúciós hálót "fixed feature extractor"-nak használjuk, azaz csak az utolsó, osztályozó réteget módosítjuk, illetve tanítjuk, a többi réteg súlyait nem változtatjuk. Ekkor az eredeti háló által tanult jellemzőket próbáljuk más osztályokra is érvényesíteni. [38] A másik esetben a hálót finomhangoljuk ("fine-tuning"), azaz az összes réteget tanítjuk egy lehetőleg kicsi tanulási rátával, hogy a modellt teljesen specializáljuk a saját adathalmazunkra. A két módszer között létezik átmenet, hogy a háló rétegeinek egy bizonyos hányadát tanítjuk újra a hátsó rétegekkel kezdve. Az általam vizsgált adathalmaz és az előtanított súlyokat meghatározó ImageNet különbözősége miatt én az utóbbi, finomhangolási módszerrel próbálkoztam, illetve

[1] Az ImageNet hivatalos oldala: `http://www.image-net.org/index`

bevezettem egy paramétert, ami megadja, hogy az előtanított rétegek mekkora hányadát finomhangoljuk, mindezt a háló végétől véve, azaz egy 0.8-as érték esetén, a modell rétegeinek első 20%-ának súlyait "befagyasztottam", míg a hátsó 80%-ot finomhangoltam.

6.2. A modell felépítése



6.1. ábra. A modellem felépítése.

6.3. A felhasznált konvolúciós modellek

A felhasznált előtanított neurális hálókat mind a PyTorch "torchvision.models"[2] moduljából importáltam, a használt rétegeket pedig a "torch.nn"[3] modulból.

6.3.1. ResNet

A ResNet [12] az egyik legnépszerűbb konvolúciós hálózat, amely áttörést hozott abban, hogy a reziduális kapcsolataival lehetővé tette a nagyon mély, sok rétegű hálózatok tanítását, kiküszöbölve az eltűnő gradiens problémát. A ResNet architektúrája tartalmaz "batch normalization" [17] rétegeket minden konvolúciós réteg után, aminek lényege, hogy az éppen átfutó "mini-batch"-et leskálázza és normalizálja, ami gyorsabbá és stabilabbá teszi a tanulást. A háló ReLU aktivációt használ. A feladat során a 18 réteg mély ResNet18-at ("torchvision.models.resnet18") alkalmaztam.

6.3.2. DenseNet

A DenseNet [15] egy olyan konvolúciós háló, amely "DenseBlock"-okból áll, amelyek lényege, hogy olyan reziduális kapcsolatok jellemzik, hogy minden réteg bemenete megegyezik az előtte levő rétegek kimenetének konkatenációival. Így minden réteg megkapja az eddig kinyert összes jellemzőt. Ez a háló is tartalmaz "batch normalization" rétegeket, és ez is ReLU aktivációt használ. A feladat során a 121 réteg mély DenseNet121-et ("torchvision.models.densenet121") használtam.

6.3.3. SqueezeNet

A SqueezeNet [16] a három háló közül a legkisebb, a legkevesebb paraméterrel rendelkezik, de ennek ellenére korrekt eredmények érhetőek el vele, ezért választottam ki ezt az architektúrát is kipróbálásra. A megalkotásának célja FPGA-kon és mobil eszközökön való használat volt. Különlegessége, hogy a paraméterszám csökkentése végett 1x1-es konvolúciós kernelekkel dolgozik, illetve alacsonyan tartja a "feature-map"-ek csatornaszámát. Ez a háló is ReLU-t használ, "batch normalization"-t azonban nem. A feladat során a SqueezeNet1.1-et alkalmaztam ("torchvision.models.squeezenet1_1").

6.4. Kapcsolat a konvolúciós és a visszacsatolt háló között

Az eredetileg betöltött konvolúciós háló utolsó, osztályozó rétegét minden esetben eldobtam, és egy adaptív maximum pooling réteget ("nn.AdaptiveMaxPool2d") tettem utána, ami a háló kimenetét egy sorvektorrá alakítja. Azért adaptív, mert a különböző konvolúciós hálók kimenetének méretei különböznek. Ezután három lineáris ("nn.Linear","fully-connected") réteget kötöttem, közöttük "dropout"-tal ("nn.Dropout") és ReLU ("nn.ReLU") aktivációval. Az első kettő réteg a vektorok

[2] A "torchvision.models" hivatalos oldala: `https://pytorch.org/docs/stable/torchvision/models.html`

[3] A "torch.nn" hivatalos oldala: `https://pytorch.org/docs/stable/nn.html`

hosszát felezi, majd a harmadik réteg kimenetének hossza egy hiperparaméter ("embedding dimension"). Az idősorban található összes kép átmegy a konvolúciós hálón, és a lineáris rétegeken, így előáll egy idősornyi "embedding dimension" hosszúságú vektor. Ezek a vektorok lesznek az első visszacsatolt réteg bemenetei.

6.5. A háló visszacsatolt rétegei

A háló visszacsatolt rétegei vagy LSTM ("nn.LSTM"), vagy GRU ("nn.GRU") rétegek. Ezek hiperparaméterei a rejtett vektor hossza, a rétegek száma, és a közöttük levő "dropout" valószínűsége. A rejtett vektor hossza megegyezik a réteg kimenetének hosszával, a rétegek száma pedig a háló ezen részének mélysége.

6.6. Osztályozó réteg

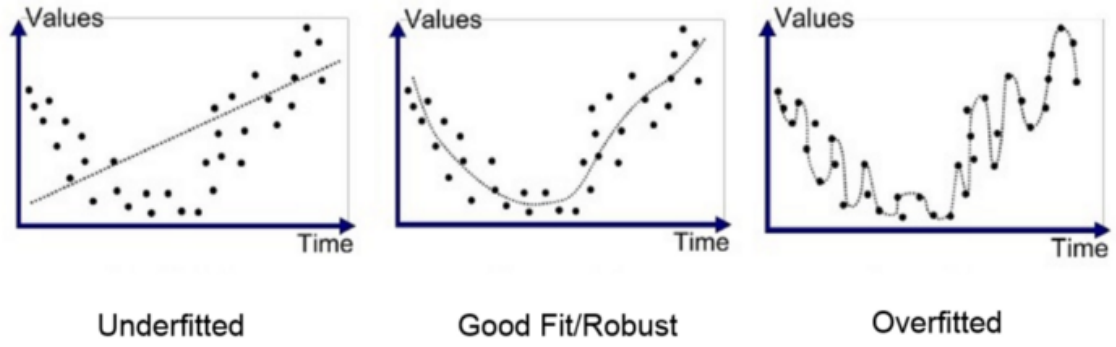
Az osztályozó réteg bemenete az utolsó rekurrens réteg kimenete. Ez az osztályozó egy lineáris rétegből áll, aminek kimenete egy olyan vektor, aminek hossza az osztályok számával egyezik meg. Ez a vektor lesz a hibafüggvény bemenete.

7. fejezet

A tanítás menetének részletei

7.1. A túltanulás

Tanítás közben a tanítás menetének két lehetséges szélsőértéke az alultanulás ("underfitting"), illetve a túltanulás ("overfitting"). Alultanulás esetén a modell tanítóadatokon vett pontossága nem nő, a modell nem tanul megfelelően. Ez akkor következik be, ha az adathalmazunk nagyon nagy, vagy a háló architektúrája, vagy valamelyik hiperparamétere egyáltalán nem megfelelő, például nagyon alacsony a tanulási ráta.



7.1. ábra. Alultanult modell, generalizálni képes modell és túltanult modell egy kétváltozós adathalmazon szemléltetve. Forrás: [2]

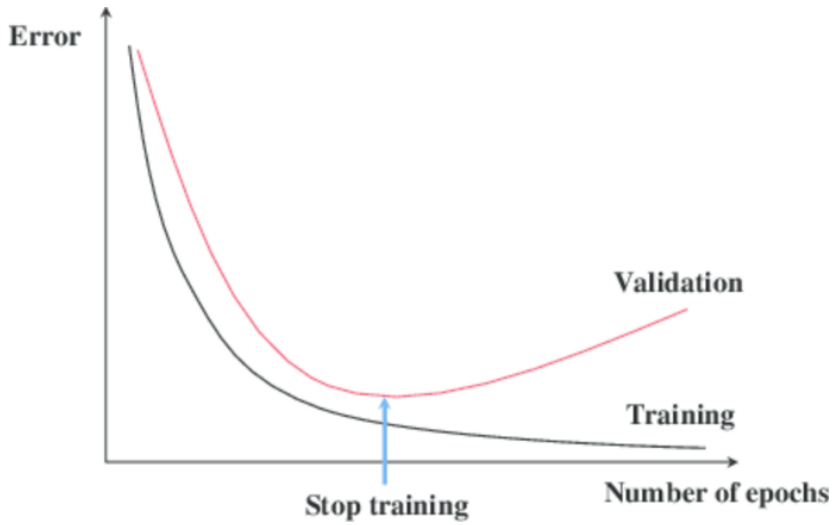
A másik eset a túltanulás, amikor a tanító adathalmazt teljesen megtanulja a háló, viszont a validációs pontosság ezzel párhuzamosan csökken. Ekkor a modell generalizációs képessége is csökken, szinte csak a tanítóadatokra lesz képes megfelelően prediktálni. Mivel pont a neurális hálók általánosítási képességét szeretnénk minél jobban növelni, ezért ez is elkerülendő. Mivel én egy kis elemszámú adathalmazzal dolgoztam, ezért főleg a túltanulás okozott problémát a feladat megoldása során, de ennek mérséklésére több módszer is létezik, ezeket összességében regularizációnak nevezik. A következőekben bemutatom azokat, amiket a feladat megoldása során alkalmaztam.

7.1.1. Dropout

A "dropout" [7] egy olyan regularizációs eljárás, aminek során a mély neurális háló minden rétegének csomópontjait, neuronjait egyenként, egymástól függetlenül egy bizonyos, hiperparaméterrel meghatározott valószínűséggel "kivesszük a hálóból", azaz a kimeneteiket kinullázzuk, megszorozzuk 0-val. Így az adott iterációban ezen neuronok súlyai nem változnak, mindig a neuronok egy különböző részhalmaza, azaz a háló egy alhálója tanul. A módszer csak a tanító szakaszban alkalmazandó, validáció során nem, ezért ekkor a predikciót vehetjük az eddig tanított alhálók közös kimenetének. Célja, hogy a tanító adathalmaz mintáiban található specifikus jellemzők megtanulását mérsékelje, így növelve a modell generalizáló képességét.

7.1.2. Early stopping

Az "early stopping", "korai megállás" lényege, hogy amikor a modell túltanul, és az eddig csökkenő validációs hiba nőni kezd (vagy valamelyik másik kiválasztott metrika értéke romlik), párhuzamosan a tanítási hiba csökkenésével, akkor egy bizonyos türelmi tanítási ciklusszám ("patience") után a tanítást befejezzük. Fontos, hogy az "early stopping" alkalmazása esetén implementálni kell egy "checkpointing" rendszert, ami minden epoch után megvizsgál egy bizonyos validációs metrikát (például a hibafüggvény értékét, vagy a validációs pontosságot), és amennyiben a háló az eddigieknél jobb értéket produkál, akkor a háló aktuális súlyait elmenti, majd a leállás után ezeket visszatölti a modellbe, és a teszt metrikák meghatározása már ezekkel történik. A hálók tanítása során, bár az F_1 érték maximálázására törekedtem, ezért célszerű lett volna ennek a metrikának a növekedésének megállásakor leállítani a tanítást, azonban a rendelkezésre álló adathalmaz alacsony elemszáma miatt ez sokszor stagnált olyan esetben is, amikor a validációs hiba még csökkent, megtartva a lehetőségét annak, hogy még javuljon a modell teljesítménye. Ezért a validációs hibához kötöttem az "early stopping" implementációmat. A "checkpointing" során viszont a legjobb F_1 értéket eredményező súlyokat mentettem el. A modellek tanítása során egységesen 25-ös "patience"-t alkalmaztam.



7.2. ábra. "Early stopping", a hibafüggvény értékének, és az "epoch"-ok számának függvényében. Forrás:[6]

7.2. Használt költség függvények

7.2.1. Categorical Cross-Entropy Loss

Először a klasszifikációs feladatok neurális hálókkal való megoldásánál általánosan használt "Categorical Cross-Entropy" hibafüggvényt ("nn.CrossEntropyLoss") alkalmaztam, ami a keresztentrópia hibafüggvény kettő vagy több osztály estén. Ennek a bemenete egy vektor, aminek hossza egyenlő a lehetséges osztályok számával, elemei pedig annak a valószínűségei, hogy az adott minta egy adott osztályba tartozik. Ez úgy keletkezik, hogy a háló utolsó rétegének kimenetére alkalmazzuk az úgynevezett "softmax" függvényt, ami azt egy 1 összegű valószínűségi eloszlásá alakítja ("output"). Az elvárt kimenet ("target") egy "one-hot" vektor, azaz egy olyan vektor, aminek csak az egyik eleme 1, minden más 0. Itt az 1-es jelöli az elvárt osztályt. A hibafüggvény a "target" vektort összeszorozza az "output" vekor negatív természetes alapú logaritmusával. Mivel a "target" vektor a valódi osztály helyén kívül mindenhol 0, ezért elég a valódi osztály helyén levő valószínűség negatív logaritmusát visszaadni.

$$s = softmax(\mathbf{o})_{\mathrm{p}} = \frac{e^{\mathbf{o}_{\mathrm{p}}}}{\sum_i^C e^{\mathbf{o}_{\mathrm{i}}}}$$

$$CE(s) = -\ln(s)$$

7.3. ábra. "Categorical Cross-Entropy Loss", ahol $\mathbf{o}$ az "output" vektor, $\mathrm{o_p}$ pedig a valódi osztályra prediktált valószínűség.

A PyTorch implementációjában megadhatjuk, hogy az egyes osztályok esetén a hiba különböző súlyozásokkal kerüljön kiszámításra. Ezt úgy tettem meg, hogy az osztályok előfordulási valószínűségeinek alapján számított, azokkal fordítottan arányos súlyokat átadtam az implementált Python osztálynak, így a ritkább osztályokba tartozó minták hibája nagyobb lesz a többihez képest, jobban befolyásolják a tanulást, így a "bias" mérsékelhető.

7.2.2. Focal Loss

A "Focal Loss" [24] a keresztentrópia hibafüggvény kiegészítése. Lényege, hogy a már helyesen klasszifikált esetek hibáját csökkenti a helytelenül klasszifikált esetekhez képest. Azaz a "nehezebb", gyakran félreosztályozott minták hibája relatíve jelentősen nagyobb lesz, mint a "könnyű", helyesen klasszifikált mintáké. Ez a kiegyensúlyozatlan adathalmaz problémáján is segíthet, hiszen így kevésbé lesz jellemző, hogy a modell minden esetben a többségi osztályt prediktálja, hiszen a helytelenül osztályozott kisebbségi osztályú minták hibája nagyobb lesz. Ezt úgy érik el, hogy egy új tényezővel megszorozzák a "Categorical Cross-Entropy" eredményét. Ez a tényező egyből kivonva a kimeneti vektor azon eleme, ami a helyes osztály valószínűségét tartalmazza, és ez a különbség a γ-dik hatványra emelve, ami a figyelem ("focus") paramétert jelöli, aminek az eredeti cikk szerint 2 az optimális értéke. Továbbá al-

kalmazható egy α súlyozó tényező is, azonban én ezt nem használtam (1 értéket adtam neki).

$$FocalLoss(s) = -\alpha(1-s)^{\gamma}\ln(s)$$

7.4. ábra. "Focal Loss", ahol s megegyezik a 3.2-es ábrán levővel.

A kódomban használt implementációt a PyTorch ökoszisztémába tartozó Kornia[1] számítógépes látással foglalkozó függvénykönyvtárból importáltam ("kornia.losses.focal.FocalLoss"), és γ-nak 2-t adtam meg.

7.3. Használt optimalizáló algoritmusok

Főleg négy optimalizáló algoritmussal kísérleteztem a feladat megoldása során, az "SGD"-vel, "RMSprop"-pal, az "Adam"-mel, illetve az "AMSGrad"-dal. Ezeket mind a PyTorch "torch.optim"[2] könyvtárából importáltam. Mindegyik adaptív algoritmus, ami azt jelenti, hogy a tényező, amivel a gradienst megszorozzuk a súlyfrissítés során (tanulási ráta), függ az eddigi gradiensek értékétől.

7.3.1. SGD

Az "SGD" (Stohasztikus Gradiens módszer) ("optim.SGD") optimalizáló algoritmusról már írtam, a gyakorlatban itt ez "mini-batch" gradiens módszert jelent. Ezt az algoritmust momentummal használtam. A momentum lényege, hogy ha a hibafüggvény minimumának keresésére úgy tekintünk, mintha egy golyó egy felületen gurulna, akkor a golyónak tehetetlenséget is tulajdonítunk, így bizonyos lokális minimumokból a golyó lendülete (momentuma) továbbviheti a golyót. Továbbá "SGD" alkalmazásakor használtam tanulási ráta ütemezőt, mégpedig "Cosine Annealing Warm Restart" ("optim.lr_scheduler.CosineAnnealingWarmRestarts") félét. Ennek lényege, hogy egy kezdeti tanulási ráta az epochok egy koszinuszos függvénye mentén lecsökken egy megadott minimum tanulási rátára, majd visszaugrik a kezdeti értékre, és ezt ciklikusan ismétli a tanulás végéig. Ez a módszer Loshchilov et al szerint kifejezetten jó eredményeket produkálhat. [25]

7.3.2. RMSprop

Az "RMSprop" ("optim.RMSprop") opimizáló algoritmust Geoffrey Hinton a Torontoi Egyetemen tartott kurzusán javasolta. [31] A kezdeti tanulási rátát minden "mini-batch" esetén a gradiensek négyzeteinek exponenciális mozgóátlagának gyökével osztjuk. A mozgóátlag paraméterét az alapértelmezett 0.9 értéken hagytam.

[1] A Kornia hivatalos oldala `https://kornia.readthedocs.io/en/latest/index.html`

[2] A "torch.optim" hivatalos oldala: `https://pytorch.org/docs/stable/optim.html`

7.3.3. Adam

Az "Adam" [19] ("optim.Adam") az "RMSprop"-hoz nagyon hasonló optimalizáló algoritmus, de itt súlyváltozás mértékének számításához a gradienst az eddigi gradiensek exponenciális mozgóátlagára frissítjük, így a momentum hatást elérve az algoritmus során, mert az eddigi gradiensek hatással vannak a jelenlegi súlyfrissítésre is. A két különböző mozgóátlag (gradiens négyzetek, gradiensek) paramétereit az alapértelmezetten hagytam (0.9, 0.99).

7.3.4. AMSGrad

Az "AMSGrad" [30] ("optim.AdamW" az "amsgrad flag" beállításával) optimalizáló algoritmus annyiban különbözik az "Adam"-tól, hogy nem az aktuális gradiens négyzetek exponenciális mozgóátlagával osztjuk a tanulási rátát, hanem ennek az értéknek az eddigi maximumával. Ez olyan esetekben, amikor az optimális minimum irányába mutató, nagyobb méretű gradienseket csak ritkán előforduló "mini-batch"-ek produkálnak, optimálisabb konvergenciát képes elérni, mint az "Adam", ami esetében ezek a ritka gradiensek az exponenciális mozgóátlag művelete során eljelentéktelenedhetnek.

7.4. Hiperparaméter optimalizáció

A feladat megoldása során a hiperparaméterek száma miatt megkíséreltem egy hiperparaméter optimalizáló keretrendszer segítségével megkeresni a legjobb kombinációkat, vagy legalább szűkíteni a paraméterteret. Az optimalizáció során a modell F1-értékének növelése volt a célom, azt adtam meg vizsgálandó metrikának. A hiperparaméter optimalizáció nehézsége a felfedezés ("exploration") illetve a kihasználás ("exploitation") közötti egyensúly megtalálása, azaz a paramétertér minél nagyobb részének felfedezése úgy, hogy az ígéretes konfigurációk kifejtésére is maradjon erőforrás (például idő, számítási kapacitás). A célra a "Ray Tune"[3] Python könyvtárat használtam. Ezen belül az "ASHA"[23] ("Asynchronous Successive Halving Algorithm") algoritmus implementációjával ("tune.schedulers.ASHAScheduler") kísérleteztem. Ennek lényege, hogy azon hiperparaméter konfigurációk, amik kevésbé ígéretesek, azokat agresszívan, hamar leállítja, hogy az erőforrások nagyobb része jusson a valószínűleg majd jobb eredményt hozó modell tanítására. Ezt úgy éri el, hogy a párhuzamos próbálkozások legjobb $1/\mu$ hányadát megtartja, majd ezekre koncentrál, és ezt iteratívan folytatja, amíg egy konfiguráció nem marad. Nálam ez μ a 4 értéket kapta. A hiperparaméterek kiválasztását a paramétertérből a "Ray Tune" "AxSearch" ("tune.suggest.ax.AxSearch") nevezetű hiperparaméterkereső algoritmusa végezte, ami az "Ax"[4] nevezetű, a gépi tanulási folyamatok automatizálását megkönnyítő könyvtárat importálja. Ennek a könyvtárnak a hiperparaméter optimalizációval foglalkozó moduljai a "BoTorch"[5][3] nevű bayes-i, és bandita optimalizáló algoritmust megvalósító könyvtárat használják. A bayes-i optimalizálás

[3] A "Ray Tune" hivatalos oldala, dokumentációja `https://docs.ray.io/en/latest/tune/index.html`

[4] Az "Ax" hivatalos oldala, dokumentációja `https://ax.dev/`

[5] A "BoTorch" hivatalos oldala, dokumentációja `https://botorch.org/`

folytonos változókra alkalmazható algoritmus, a bandita algoritmus pedig diszkrét változókra. Az "Ax" és a "BoTorch" egyaránt a Facebook Open Source terméke.

	Hiperparaméter lehetséges értékei
Konvolúciós háló típusa	*Resnet*, *DenseNet*, *SqueezeNet*
Konvolúciós háló tanított hátsó hányada	1, 0.8, 0.6
Konvolúciós háló dropout	0.3, 0.5, 0.7
Embedding dimension értéke	32, 64, 128
Visszacsatolt háló típusa	*LSTM*, *GRU*
Visszacsatolt háló mélysége	2, 3
Visszacsatolt háló rejtett vektorának hossza	32, 64, 128
Visszacsatolt háló dropout	0.3, 0.5, 0.7
Optimalizáló algoritmus	*RMSprop*, *Adam*, *AMSGrad*
Batch méret	4, 8, 16
Tanulási ráta	$[1e-5, 1e-3]$

7.1. táblázat. A hiperparaméter optimalizálás paramétertere.

Az implementációban beállítottam "random seed"-et ("torch.manual_seed", "numpy.random.seed"), így a kísérletek reprodukálhatóak.

A hiperparaméter optimalizáció során minden használt konvolúciós hálóra külön folyamatot indítottam, ami egyenként 50 paraméter konfigurációváltozat kipróbálását tartalmazta, maximum 20 tanítási ciklus erejéig. Ez több órát vett igénybe, illetve különlegesen hosszú időbe telt, körülbelül 10 órába, a DenseNet esetén, ami a legszámításigényesebb a három felhasznált konvolúciós háló közül. A folyamat során az "SGD" optimalizáló algoritmust nem használtam, az csak nagyon lassan konvergált optimális eredményre. Azonban amikor megkaptam a legjobb eredményt produkáló hiperparaméter konfigurációkat, akkor azt felhasználva, az előzőleg bemutatott "SGD" módszert alkalmazva sikerült jobb eredményeket elérnem a SqueezeNet architektúra esetén (5%-al magasabb F_1 érték, mint a 7.2.-es táblázatban). Viszont ez lényegesen több tanulási ciklust igényelt.

A hiperparaméter optimalizáció során végig a "Focal Loss"-t alkalmaztam hibafüggvénynek, illetve eközben túlmintavételezést is alkalmaztam. Továbbá minden esetben a három osztályba csoportosítás feladatát optimalizáltam, mivel a folyamatot mindkét klasszifikációs módszerre, és minden CNN típusra futtatni nagyon hosszú időbe telt volna, ezért az általam nehezebb feladatnak vélt osztályozásra koncentráltam, azt feltételezve, hogy egy olyan konfiguráció, ami a nehezebb feladaton jól teljesít (nehezebb, hiszen a nem egészséges páciensek esetében is hipertrófia, azaz vastagabb szívfal áll fenn, de ebből külön kell vennie a HCM-es mintákat is), az a könnyebbet is képes abszolválni. De elképzelhető, hogy ha a két osztály feladatára optimalizáltam volna, akkor találok optimálisabb hiperparaméter konfigurációt.

Az idősorként értelmezhető adathalmazokon végzett tanítás egy további fontos hiperparamétere az időablak hossza, amit mint egyedi mintát vizsgálunk, de feltételezem, hogy jobb minél hosszabb összefüggő idősorral dolgozni, amennyiben az belefér a GPU memóriájába. Az én esetemben a 13 hosszú adatsorok nem okoztak problémát ebből a szempontból, ezért egyben használtam ezeket.

	ResNet	DenseNet	SqueezeNet
CNN tanított hányada	1.0	0.6	0.8
CNN dropout	0.3	0.3	0.3
Embedding dimension	128	64	128
RNN típusa	LSTM	LSTM	LSTM
RNN mélysége	2	2	2
RNN rejtett vektor hossza	64	64	64
RNN dropout	0.5	0.5	0.7
Optimalizáló alg	RMSprop	RMSprop	AMSGrad
Batch méret	16	8	8
Tanulási ráta	4.5e-05	5e-05	8.5e-05
Teszt pontosság	0.7857	0.6905	0.6429
Teszt F1-érték	0.7454	0.6743	0.6462

7.2. táblázat. A hiperparaméter optimalizálás eredménye, a talált optimális konfiguráció három osztály esetén.

Látszik, hogy több hiperparaméter mindhárom konvolúciós háló architektúra esetében ugyanazt az értéket vette fel. Ilyen pédául a visszacsatolt háló típusa (LSTM), és rejtett vektorának hossza (64), illetve a konvolúciós háló esetében a "dropout" értéke (0.3). Az is megfigyelhető, hogy ebben a három osztályba soroló módszernél az önálló laboratórium során produkált eredményeket (8.2-es táblázat) felülmúlták a szív időbeli mozgását is felhasználó, jelenlegi modellek.

Mivel a ResNet alapú modellel sikerült a legjobb eredményt elérnem, ezért ennek az optimálisnak talált konfigurációját értékelem ki részletesebben, egyes hiperparaméterek értékét megváltoztatva, majd a kapott eredményeiket összehasonlítva.

8. fejezet

Eredmények összehasonlítása

8.1. Önálló laboratórium előzmények

Az Önálló laboratórium nevű tárgyat is a balkamrai-hipertrófia osztályozás témakörében végeztem. Ennek során a szív térbeli és időbeli kiterjedésének információtartalmát nem használtam fel, a páciensek MRI felvételeinek különböző "slice"-ait egymástól függetlenül adtam be egy konvolúciós hálóba. Csak a "systole", illetve "dyastole" "frame"-eit használtam, a modell nem tartalmazott visszacsatolt rétegeket, illetve nem egy páciens egy "frame"-ének a három "slice"-a volt a bemenet, mint háromcsatornás kép, hanem mindegyik szeletet háromszor egymás után tettem, hogy meglegyen a három csatorna.

Az önálló labor feladat alatt ugyanezzel a három előtanított hálóval próbálkoztam. A tárgy során elért eredmények az alábbiak:

	Resnet18	DenseNet121	SqueezeNet1.1
Test Accuracy	0.6965	0.7189	0.6542
F1 score	0.7015	0.7205	0.6634

8.1. táblázat. Az Önálló laboratórium eredményeim két osztály esetén.

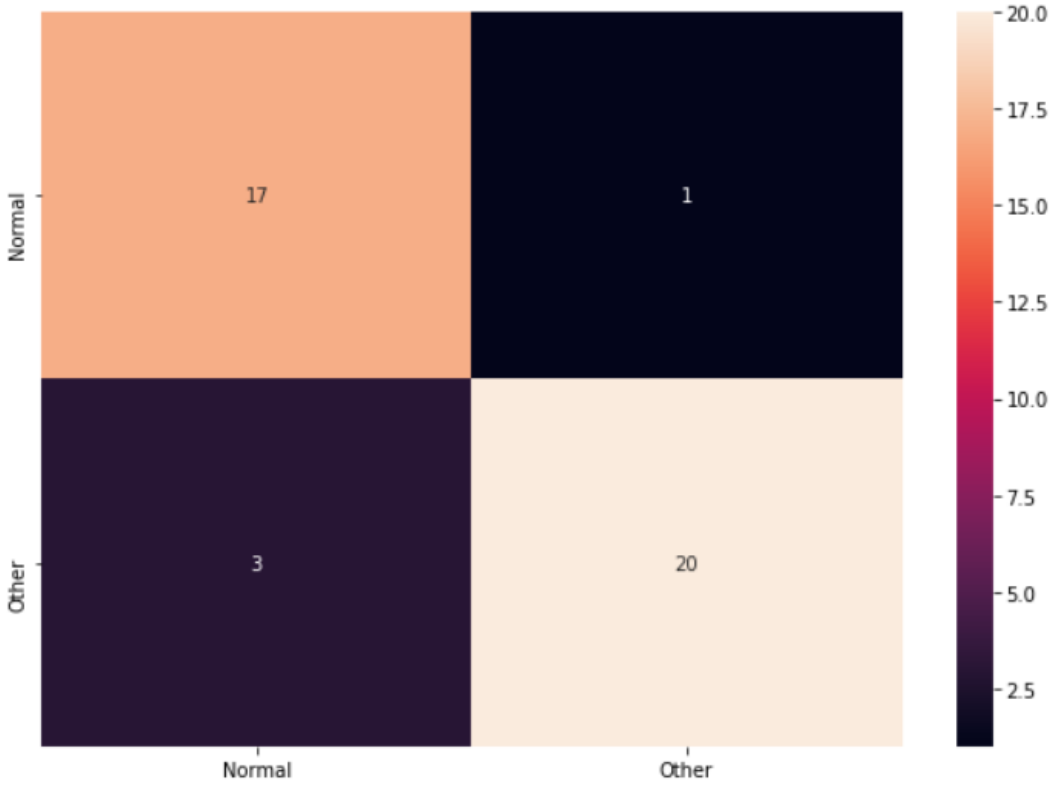
	Resnet18	DenseNet121	SqueezeNet1.1
Test Accuracy	0.6616	0.6741	0.6417
F1 score	0.6584	0.6737	0.6353

8.2. táblázat. Az Önálló laboratórium eredményeim három osztály esetén.

8.2. A két osztályra bontás feladata

	Resnet18	DenseNet121	SqueezeNet1.1
Test Accuracy	0.8780	0.805	0.9024
F1 Score	0.8785	0.805	0.9027

8.3. táblázat. A szakdolgozat eredményeim két osztály esetén.



8.1. ábra. A két osztályra bontás feladattában a legjobban teljesítő SqueezeNet tévesztési mátrixa.

A két osztályra bontás feladatában érdekes módon a SqueezeNet alapú modell teljesített a legjobban a hiperparaméter optimalizáció során talált konfigurációk felhasználásával. Elképzelhető, hogy a kevesebb tanítható paraméter miatt kevésbé tanult túl, mint a nagyobb hálók, így ezen a három osztályra bontásnál kevésbé komplex feladat során képes volt jobban generalizálni. Az látszik, hogy ennél az osztályozási módszernél az önálló laboratórium során produkált eredményeket (8.1-es táblázat) felülmúlták a szív időbeli mozgásának információtartalmát is felhasználó modellek.

8.3. A három osztályra bontás feladata

Az alábbiakban bemutatok néhány esetet, hogy bizonyos, a tanítás módszerében megmutatkozó különbségek hogyan hatnak az eredményekre. Alapértelmezetten a

hiperparaméter optimalizáció során a legjobb eredményeket produkáló ResNet alapú modellt használom itt, az akkor használt tanítási módszerekkel (kivágott képeken tanítás, túlmintavételezés, "FocalLoss"), és az optimálisnak talált hiperparaméter konfigurációval. Az ettől való eltérést jelzem.

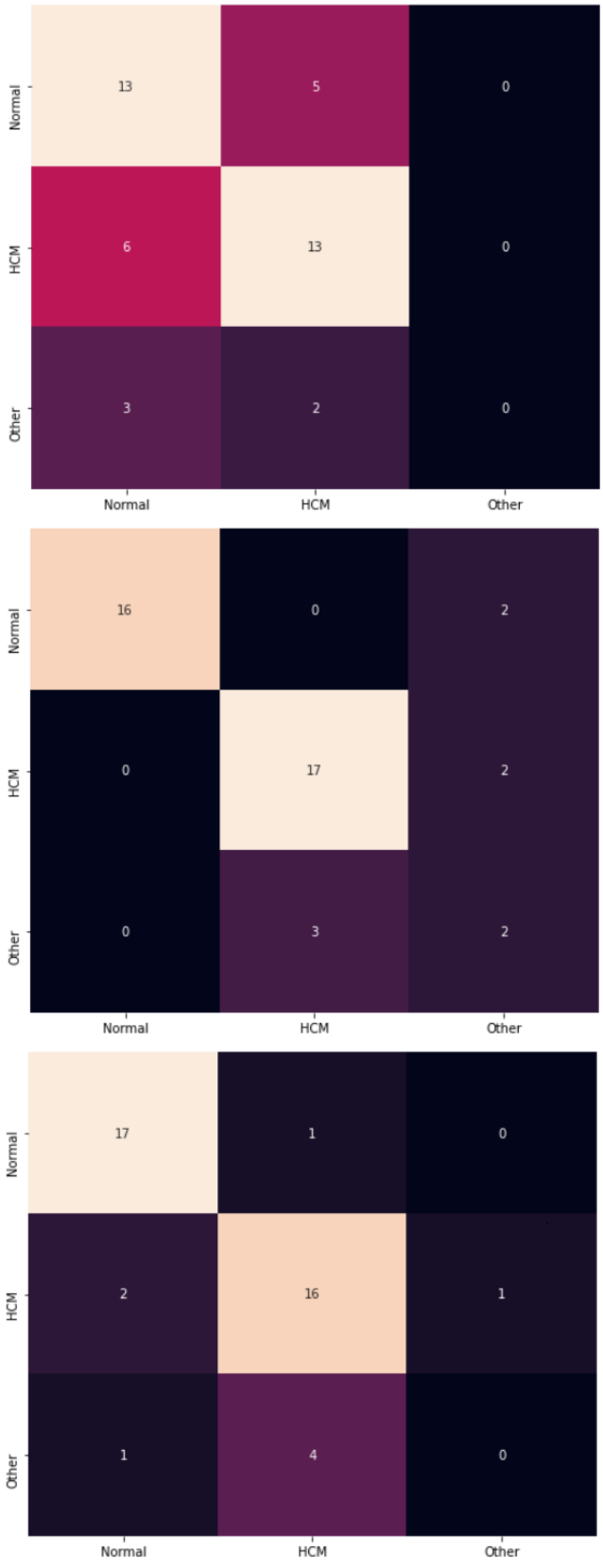
8.3.1. Az adathalmaz kiegyensúlyozására tett kísérletek

	CrossEntropy-Loss	súlyozott Cross-EntropyLoss	Oversampling + FocalLoss
Test Accuracy	0.619	0.833	0.786
F1 Score	0.580	0.841	0.745

8.4. táblázat. Az adathalmaz kiegyensúlyozására tett kísérletek összehasonlítása három osztáy esetén

Összehasonlítottam azt az esetet, amikor semmilyen módszerrel nem próbálom az adathalmaz kiegyensúlyozatlanságát kompenzálni, azzal amikor ezt az osztályok előfordulásának megfelelően súlyozom a CrossEntropyLoss által visszaadott hibát, illetve azt, amikor túlmintavételezést ("oversampling") és "FocalLoss"-t egyaránt alkalmazok. Ezt a három osztályba sorolás feladatán teszem meg, mert két osztály esetén a kiegyensúlyozatlanság csekély mértékű, azonban három osztály esetén van egy kisebbségi osztály, aminek mintáinak darabszáma körülbelül a negyede a másik két osztály mintáinak számának. Az eredményeket megvizsgálva azt tapasztaltam, hogy a ResNet-et és a hozzá talált hiperparaméter konfigurációt használva a súlyozott CrossEntropyLoss produkálta legjobb eredményt.

A 8.2.-es ábrán található tévesztési mátrixokon az látható, hogy a legjobb metrikákat adó módszer és a második helyezett között annyi a különbség, hogy a "pontosabb" néhány mintát a kisebbségi osztályba sorolt, amik közül kettő predikció jó lett, de a modell így is elfogult ("biased") volt a több mintával rendelkező osztályok irányába Az is megfigyelhető, hogy amikor nem alkalmaztam semmilyen módszert, akkor egyetlen egy mintát sem prediktált a modell a kisebbségi osztályba.



8.2. ábra. A kiegyensúlyozási módszerek tévesztési mátrixokon, CrossEntropyLoss (fent), súlyozott CrossEntropyLoss (középen), Oversampling + FocalLoss (lent).

8.3.2. A teljes képeken való tanítás

	Kivágott képek	Teljes képek
Test Accuracy	0.7857	0.5476
F1 Score	0.7454	0.5461

8.5. táblázat. A képek kivágott, lényegi részén való tanítás összehasonlítása a teljes képeken való tanítással három osztály esetén.

Az eredményekből látszik, hogy a kivágott képeken való tanítás eredménye lényegesen jobb, mint a teljes képeken való tanítás. Ennek oka az lehet, hogy a teljes képek területének körülbelül $\frac{3}{4}$-e a feladat megoldása szempontjából lényegtelen zajt tartalmazott, ezzel megnehezítve a tanítás menetét, "underfitting"-et okozva.

8.3.3. Előtanított háló és véletlenszerűen inicializált súlyok

	Előtanított súlyok	Random inicializált súlyok
Test Accuracy	0.7857	0.6428
F1 Score	0.7454	0.6499
Epoch-ok a megállásig	35	50

8.6. táblázat. A modell súlyainak inicializálásának hatása az eredményekre.

Látszik, hogy Tajbakhsh et al [37] állítását sikerült megerősítenem, valóban kevesebb epoch alatt konvergál, és jobb eredményt produkál egy előtanított háló a véletlenszerűen inicializált súlyokhoz képest, még akkor is, amikor az adathalmaz, amin az előtanítást végezték, teljesen különböző jellegű ahhoz képest, amin a "pretrained" hálót alkalmaztam.

8.4. Összefoglalás, kitekintés

Szakdolgozatomban szív MRI felvételeket osztályoztam balkamrai-hipertrófia szempontjából. Ehhez felhasználtam a szív mozgásában található információt is, ezért a feladatot ismert, előtanított konvolúciós hálók és visszacsatolt hálók kombinálásával oldottam meg. Két csoportosítást alkalmaztam, először két osztályba (normális, vagy beteg), majd három osztályba (normális, HCM-es, egyéb betegség). Ugyanezzel a feladattal az Önálló laboratórium tárgy keretein belül, az adatok időbeliségét nem felhasználva már foglalkoztam.

Az elvárásaimnak megfelelően a szív mozgásából kinyerhető információk növelték a modell teljesítményét, mind a két osztályozási feladat esetén sikerült az eddigi eredményeket felülmulni. Azonban a kiegyenlítettlen adathalmaz okozta problémákat csak részben sikerült kiküszöbölnöm.

A modell teljesítményét elsősorban egy nagyobb, kiegyenlítettebb adathalmazzal lehetne javítani, illetve elképzelhető, hogy a visszacsatolt alháló komplexitásának növelésével is jobb eredményeket produkálna a háló. Összességében egyetértek Bernard et al-lal [4] abban, hogy a hasonló megközelítésű, mély neurális hálókon alapuló orvosi diagnosztikai kiegészítéseknek a közeljövőben nagy szerepe lesz az automatizált vizsgálati folyamatokban, ezzel megkönnyítve az orvosok munkáját.

Irodalomjegyzék

[1] Aktivációs függvények ábra, 2020. december.
URL https://docs.paperspace.com/machine-learning/wiki/activation-function.

[2] Alul- és túltanulás ábra, 2020. december. URL https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76.

[3] Maximilian Balandat – Brian Karrer – Daniel R. Jiang – Samuel Daulton – Benjamin Letham – Andrew Gordon Wilson – Eytan Bakshy: BoTorch: Programmable Bayesian Optimization in PyTorch. *arXiv e-prints*, 2019. október., arXiv:1910.06403. p.

[4] Olivier Bernard – Alain Lalande – Clement Zotti – Frederic Cervenansky – Xin Yang – Pheng-Ann Heng – Irem Cetin – Karim Lekadir – Oscar Camara – Miguel Angel Gonzalez Ballester – Gerard Sanroma – Sandy Napel – Steffen Petersen – Georgios Tziritas – Elias Grinias – Mahendra Khened – Varghese Alex Kollerathu – Ganapathy Krishnamurthi – Marc-Michel Rohé – Xavier Pennec – Maxime Sermesant – Fabian Isensee – Paul Jager – Klaus H Maier-Hein – Peter M. Full – Ivo Wolf – Sandy Engelhardt – Chrisitan Baumgartner – Lisa Koch – Jelmer Wolterink – Ivana Isgum – Yeonggul Jang – Yoonmi Hong – Jay Patravali – Shubham Jain – Olivier Humbert – Pierre-Marc Jodoin: Deep Learning Techniques for Automatic MRI Cardiac Multi-structures Segmentation and Diagnosis: Is the Problem Solved? *IEEE Transactions on Medical Imaging*, 37. évf. (2018. május) 11. sz., 2514–2525. p. URL https://hal.archives-ouvertes.fr/hal-01803621.

[5] Kyunghyun Cho – Bart van Merriënboer – Caglar Gulcehre – Dzmitry Bahdanau – Fethi Bougares – Holger Schwenk – Yoshua Bengio: Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (konferenciaanyag). Doha, Qatar, 2014. október, Association for Computational Linguistics, 1724–1734. p.
URL https://www.aclweb.org/anthology/D14-1179.

[6] Early stopping ábra, 2020. december.
URL https://towardsdatascience.com/a-practical-introduction-to-early-stopping-in-machine-learning-550ac88bc8fd.

[7] Ian Goodfellow – Yoshua Bengio – Aaron Courville: *Deep Learning*. 2016, MIT Press. http://www.deeplearningbook.org.

[8] Gru ábra, 2020. december. URL https://www.data-blogger.com/wp-content/uploads/2017/08/gru.png.

[9] Hadamard-product, 2020. Nov.
URL https://en.wikipedia.org/wiki/Hadamard_product_(matrices).

[10] Guo Haixiang – Yijing Li – Jennifer Shang – Gu Mingyun – Huang Yuanyue – Bing Gong: Learning from class-imbalanced data: Review of methods and applications. *Expert Systems with Applications*, 73. évf. (2016. 12).

[11] Haibo He – E.a. Garcia: Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21. évf. (2009) 9. sz., 1263–1284. p.

[12] Kaiming He – Xiangyu Zhang – Shaoqing Ren – Jian Sun: Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[13] Sepp Hochreiter – Jürgen Schmidhuber: Long short-term memory. *Neural Computation*, 9. évf. (1997) 8. sz., 1735–1780. p.

[14] Kurt Hornik – Maxwell Stinchcombe – Halbert White: Multilayer feedforward networks are universal approximators. *Neural Networks*, 2. évf. (1989) 5. sz., 359–366. p.

[15] Gao Huang – Zhuang Liu – Laurens van der Maaten – Kilian Q. Weinberger: Densely connected convolutional networks, 2018.

[16] Forrest N. Iandola – Song Han – Matthew W. Moskewicz – Khalid Ashraf – William J. Dally – Kurt Keutzer: Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size, 2016.

[17] Sergey Ioffe – Christian Szegedy: Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[18] Robert Keim:. https://www.allaboutcircuits.com/technical-articles/how-to-perform-classification-using-a-neural-network-a-simple-perceptron-example/, 2020. december.

[19] Diederik P. Kingma – Jimmy Ba: Adam: A method for stochastic optimization, 2014.
URL http://arxiv.org/abs/1412.6980. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[20] Konvolúció ábra, 2020. december. URL https://people.minesparis.psl.eu/fabien.moutarde/ES_MachineLearning/TP_convNets/convolve.png.

[21] Konvolúciós neurális háló ábra, 2020. Dec. URL https://en.wikipedia.org/wiki/Convolutional_neural_network#/media/File:Typical_cnn.png.

[22] Y. Lecun – B. Boser – J. S. Denker – D. Henderson – R. E. Howard – W. Hubbard – L. D. Jackel: Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1. évf. (1989) 4. sz., 541–551. p.

[23] Liam Li – Kevin Jamieson – Afshin Rostamizadeh – Ekaterina Gonina – Jonathan Ben-tzur – Moritz Hardt – Benjamin Recht – Ameet Talwalkar: A system for massively parallel hyperparameter tuning. In I. Dhillon – D. Papailiopoulos – V. Sze (szerk.): *Proceedings of Machine Learning and Systems* (konferenciaanyag), 2. köt. 2020, 230–246. p. URL https://proceedings.mlsys.org/paper/2020/file/f4b9ec30ad9f68f89b29639786cb62ef-Paper.pdf.

[24] T. Lin – P. Goyal – R. Girshick – K. He – P. Dollár: Focal loss for dense object detection. In *2017 IEEE International Conference on Computer Vision (ICCV)* (konferenciaanyag). 2017, 2999–3007. p.

[25] I. Loshchilov – F. Hutter: Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations (ICLR) 2017 Conference Track* (konferenciaanyag). 2017. április.

[26] Lstm működése, 2020. december.
URL https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[27] Lstm ábra, 2020. Dec. URL https://en.wikipedia.org/wiki/Long_short-term_memory#/media/File:The_LSTM_cell.png.

[28] Magnetic resonance imaging, 2020. Nov.
URL https://en.wikipedia.org/wiki/Magnetic_resonance_imaging.

[29] P. K. Rudra Poudel – Pablo Lamata – Giovanni Montana: Recurrent fully convolutional neural networks for multi-slice mri cardiac segmentation. *RAMBO+HVSMR@MICCAI*, 2016., 83–94. p.

[30] Sashank Reddi – Satyen Kale – Sanjiv Kumar: On the convergence of adam and beyond. In *International Conference on Learning Representations* (konferenciaanyag). 2018.

[31] Rmsprop, 2020. december. URL https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[32] F. Rosenblatt: The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65. évf. (1958) 6. sz., 386–408. p.

[33] Holger R. Roth – Christopher T. Lee – Hoo-Chang Shin – Ari Seff – Lauren Kim – Jianhua Yao – Le Lu – Ronald M. Summers: Anatomy-specific classification of medical images using deep convolutional nets. *2015 IEEE 12th International Symposium on Biomedical Imaging (ISBI)*, 2015.

[34] David E. Rumelhart – Geoffrey E. Hinton – Ronald J. Williams: *Learning Representations by Back-Propagating Errors*. Cambridge, MA, USA, 1988, MIT Press, 696–699. p. ISBN 0262010976. 4 p.

[35] Iram Shahzadi – Tong Boon Tang – Fabrice Meriadeau – Abdul Quyyum: Cnn-lstm: Cascaded framework for brain tumour classification. *2018 IEEE-EMBS Conference on Biomedical Engineering and Sciences (IECBES)*, 2018.

[36] Szív, 2020. december. URL https://hu.wikipedia.org/wiki/Szív.

[37] Nima Tajbakhsh – Jae Y. Shin – Suryakanth R. Gurudu – R. Todd Hurst – Christopher B. Kendall – Michael B. Gotway – Jianming Liang: Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE Transactions on Medical Imaging*, 35. évf. (2016) 5. sz., 1299–1312. p.

[38] Transfer learning, stanford university, 2020. december.
URL https://cs231n.github.io/transfer-learning/.

[39] Ventricular hypertrophy, 2020. Oct.
URL https://en.wikipedia.org/wiki/Ventricular_hypertrophy.

[40] Visszacsatolt háló ábra, 2020. Dec. URL https://upload.wikimedia.org/wikipedia/commons/b/b5/Recurrent_neural_network_unfold.svg.

[41] Dongqing Zhang – Ilknur Icke – Belma Dogdas – Sarayu Parimal – Smita Sampath – Joseph Forbes – Ansuman Bagchi – Chih-Liang Chin – Antong Chen: A multi-level convolutional lstm model for the segmentation of left ventricle myocardium in infarcted porcine cine mr images. 2018. 11.