TANSZÉKVEZETŐ

SZAKDOLGOZAT FELADAT

Dobsinszki Gergely
mérnökinformatikus hallgató részére

Mély neuronháló alapú orosz nyelvű beszédfelismerő fejlesztése

A nagy pontosságú gépi beszédfelismerés megvalósítása régi vágya az emberiségnek. Napjaink egyik legígéretesebb megközelítése a tisztán mély-neuronháló alapú (end-to-end) technika, amely nemcsak pontosságában múlhatja felül a hagyományosabb megközelítéseket, de nyelvek közötti hordozhatósága is jelentős előnyt jelent. A hallgató feladata, hogy angol nyelvű példa alapján orosz nyelvű beszédfelismerőt hozzon létre mély neuronháló alapon, nyílt beszédadatbázisokat felhasználva.

A hallgató feladatának a következőkre kell kiterjednie:

- Végezzen irodalomkutatást a beszédfelismerési témában és mutassa be a tisztán neurális háló alapú end-to-end beszédfelismerő működési elvét.
- Elemezze a korszerű, mély neuronhálós beszédfelismerő architektúrákat és tervezzen egyet a célfeladatra, szem előtt tartva az alacsony erőforrásigényt.
- Készítsen orosz nyelvű beszédfelismerőt csak célnyelvi adatok felhasználásával, illetve vizsgálja meg az ún. transfer-learning lehetőségét angol-orosz nyelvpárra.
- Optimalizálja neurális hálózatokat és értékelje ki a felismerési eredményeket.
- Dokumentálja munkáját, végül összegezze a tapasztalatait és javasoljon további fejlesztési irányokat.

Tanszéki konzulens: Dr. Mihajlik Péter

Budapest, 2020. október 6.

/ Dr. Magyar Gábor /
tanszékvezető

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai Tanszék

1117 Budapest Magyar tudósok krt. 2.
Telefon: +36 1 463 2448 Fax: +36 1 463 3107
E-mail: magyar@tmit.bme.hu http://www.tmit.bme.hu

Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Távközlési és Médiainformatikai Tanszék

Mély neuronháló alapú orosz nyelvű beszédfelismerő fejlesztése

Szakdolgozat

Készítette

Dobsinszki Gergely

Konzulens

dr. Mihajlik Péter

2020. december 7.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Dobsinszki Gergely*, 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 7.

Dobsinszki Gergely
hallgató

Kivonat

Amióta lehetséges az emberi hang digitális rögzítése, roppantul izgatja fantáziánkat annak automatikus értelmezése, írásos formában való rögzítése. Ennek számos előnye van, többek közt egy gyors kommunikációs lehetőséget jelent a géppel, ami speciális helyzetekben igen hasznos tud lenni, mint például mentő hívása, de a mindennapokban is jelentős kényelmi szempontot jelent.

Számos implementáció létezik, mint az Amazon Echo, vagy a Microsoft Cortana személyi asszisztense, melyek a bemenő hangot értelmezve igyekeznek válaszolni kérdéseinkre vagy egyéb feladatok elvégzését végezhetik. Ezek a termékek és alkalmazások először az angol nyelvterületeken terjedtek el, aminek oka a nyelvben magában keresendő, hiszen kezdetben angol nyelvű társalgásra, bemenetre készültek, és még a mai napon csak egy maroknyi nyelven lehet kommunikálni vele. A világon rengeteg nyelv létezik és többnél is felmerül az igény, hogy létezzen hozzájuk egy működő beszédfelismerő alkalmazás. A fentiek miatt munkám során az egyik fontos, de a beszédfelismerés szempontjából elhanyagoltabb nyelvvel, az orosszal foglalkoztam.

Egy fontos eltérés a munkámban hagyományos megközelítésekhez képest, hogy neurális hálókat használ, csökkentve ezzel a felismeréshez használt elemek számát. A feladat során megvizsgáltam annak lehetőséget, hogy egy létező, orosztól eltérő nyelvmodellből kiindulva lehetséges-e jobb eredményeket elérni, mint a semmiből felépítve, és igen biztató eredményeket értem el. A jövőben több módszert is lehet alkalmazni, vagy a meglévő paramétereket optimalizálni a dekódolás pontosságának javítása végett.

Abstract

This document is a LaTeX-based skeleton for BSc/MSc theses of students at the Electrical Engineering and Informatics Faculty, Budapest University of Technology and Economics. The usage of this skeleton is optional. It has been tested with the *TeXLive* TeX implementation, and it requires the PDF-LaTeX compiler.

1. fejezet

Bevezetés

1.1. Gépi beszédfelismerés

Az automatikus gépi beszédfelismerés (ASR) régóta foglalkoztatja az emberek fantáziáját. Személyi asszisztens formájában már évtizedekkel ezelőtt találkozhatott vele az átlagos sci-fi kedvelő. De már napjainkban is ott lapul a zsebünkben a technológia, az okos-telefonjaink, melyek a legtöbb ember számára a mindennapok részét képzik, rendelkeznek beszédfelismerő funkcióval. Legyen az egy egyszerűbb személyi asszisztens, amely segítségével például könnyedén beállíthatóak emlékeztetők, vagy egy egyszerű Google Search-ön alapuló keresés, melyhez a begépelést hang segítségével is el lehet végezni. Nappalinkban is egyre inkább hódít, jó példa rá az Amazon Alexa-ja, aminek a segítségével több okos eszköz is kezelhető a lakóterünkben.

Ezek a technológiák mind automatikus beszédfelismerésen alapszanak, amelyet alkalmazva az ember kizárólag hang interfész segítségével tud kommunikálni a géppel, mintha csak egy másik emberrel csevegne. Ez a megközelítés nagy szabadságot nyújt a kommunikáló számára, nem köti többé billentyűzet, egér, érintőképernyő de még monitor, kijelző sem. Az egyre inkább pontosodó, esetenként már az emberi precizitást is megközelítik[**?**], felismerés nyújtotta lehetőségek felhasználási területe egyre bővül, a technológia egyre inkább bekerül a köztudatba és mindennapjainkba.

1.2. Út az end to end alapú gépi beszédfelismerésig

A fonográf, egy Thomas Eddison által jegyzett 1877-es találmány, volt az első eszköz mely hang rögzítésére alkalmas volt. A módszer finomításával és térhódításával, illetve a számítógépek megjelenésével, és a digitalizációval egyre inkább elterjedt és fejlődött a hangrögzítés. Az első komoly eredményeket felmutató kutatási projektet a beszédfelismerés területén az Egyesült Államokbeli Fejlett Védelmi Kutatási Projektek Ügynöksége (DARPA) finanszírozta. A kutatás célja egy 1000 szavat felismerni képes program megalkotása volt, melyet a kitűzött 5 éven belül sikerült is megalkotni [12].

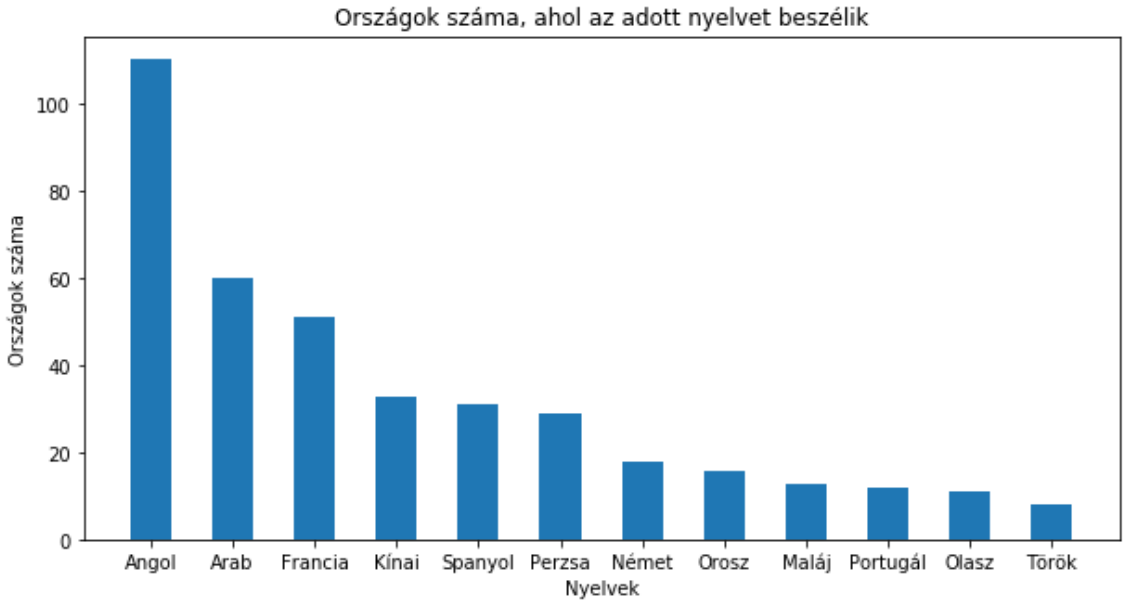
Az 1980-as években egy újabb technológia a rejtett Markov-modell (HMM), egy időbeli valószínűségi modell, forradalmasította a gépi beszédfelismerést. A modell segítségével lényegesen javítható volt a beszédfelismerés pontossága. Az HMM napjainkban is nagy sikerességnek örvend, rendkívül elterjedten használják különböző elemekkel, akusztikus- és nyelv modellekkel párosítva a minél nagyobb pontosságú beszédfelismerés végett.

A neurális hálók térhódításával 2010-es években megjelent egy alternatív, folyamatosan terjedő megközelítés, az end-to-end alapú beszédfelismerés. Segítségével kiküszöbölhető a komplex, több elemből álló módszerek alkalmazása, és egy egységből felépülő modell állítható elő. A munkám is ezen a technológian alapszik.

1.3. Az orosz nyelv

Ahogy az a 1.1-es ábrán látható világszerte az egyik legelterjedtebb nyelv az angol, mely nem csak a közösségi, társadalmi és politikai, hanem a tudományos életet is uralja. Nem meglepő, hogy ezt a nyelvet tekintjük a beszédfelismerő rendszerek pontosságának mérésekor alapnak, ezen a nyelven vannak a legkiterjedtebb szabadon elérhető beszéd adatbázisok is.

Számos nyelvet beszélnek világszerte az angol mellett, sokak számára viszont az angol nem az anyanyelvük, estleg egyáltalán nem beszélik azt. Beszédfelismerés során tehát szükséges egyéb hivatalos nyelveket is vizsgálni és kutatni.



1.1. ábra. Országok száma, melyekben az adott nyelvet beszélik. [11]

Az orosz több mint 140 millió ember anyanyelve és további 110 millióan használják az oroszt második nyelvként[1], így az orosz nyelv igen fontos szerepet tölt be a nemzetközi porondon. Az 1.1-es ábrán látható, hogy elterjedt nyelvnek számít több országban is. Az end-to-end technológia és a neurális hálók által nyújtott lehetőségek megkönnyítik a nyelvek közti átállást, így egy angoltól eltérő nyelv vizsgálata és kutatása gördülékenyen megvalósítható egy angol nyelvű példa alapján.

Munkám az orosz nyelvű beszédfelismerés lehetőségeit mutatja-be, de segítségével tetszőleges idegen nyelven, az itt bemutatott irányelvek mentén, lehetséges a beszédfelismerés kutatása és pontosítása.

1.4. Diplomaterv felépítése

A követelmények elemzése fejezetben a munkám főbb szempontjai, a sarkalatos pontok és kikötések kerülnek részletezésre, kifejtésre. Ezen pontok kidolgozásának módjai, esetleges nehézségei kerülnek említésre. Felsorakoztatom a tervezés előtt felmerült ötleteket, azok megvalósításának lehetőségeit.

Az elméleti összefoglaló rész a beszédfelismerés teóriájával foglalkozik, különösképp kitérve a neurális hálókat alkalmazó implementációra és azok alapjaira.

[1] Az orosz nyelv, Wikipedia `https://hu.wikipedia.org/wiki/Orosz_nyelv`

Továbbá tartalmazza az eddigi legjobb megoldásokat, melyeket össze is hasonlít, röviden értékelve azok sikerességét. A levonható következtetések fényében felvázolja a fejlesztés kiindulási alapjának tekinthető állapotot és tervezésbeli döntéseket.

A tervezés és fejeztet magába foglalja az előzetes döntéseket és azok indoklását, a lehetséges kísérletek várható eredményét. További lehetőségeket, ötleteket, optimalizálási lehetőségeket is felsorakoztat, hogy az implementáció résznél helyenként több opció közül lehessen választani.

Az implementáció fejezetben a választott megoldások megvalósítását írom le, röviden kitérve a mögöttes logikára, alátámasztva és indokolva azok sikerességét. Helyenként, ahol azt a további implementációs döntések megindokolják, rövid eredményeket is közlök.

Az összefoglaló tartalmazza az elért eredmények bemutatását, azok értékelését és a végső következtetések levonását, illetve a további kutatási lehetőségek felvázolását.

2. fejezet

A követelmények elemzése

2.1. Beszédfelismerés bemutatása

Az irodalomkutatás során az end-to-end alapú beszédfelismerést különböző források, cikkek, internetes blogok és papírok alapján végezni. Mivel egy egészen új és feltörekvő megközelítésről van szó, célszerű a forrásokat megvizsgálni, összevetni az azonos témáról írtakat az alátámasztás és a könnyebb értelmezhetőség végett.

A neurális hálók működésének bemutatása elengedhetetlen a szerepük nyomán a beszédfelismerésben. Fontos kitérni az end-to-end alap mivoltára, ennek előnyeinek és hátrányainak bemutatása az eddig bevált, hagyományos módszerekkel szemben.

Érdemes az elméleten túl a potenciálisan felhasználható megoldásokat részletesen megismerni, melyek implementálhatók a későbbi tervezés során a jobb eredmények végett.

2.2. Korszerű architektúrák és tervezés

Többféle neuronhálós architektúrák léteznek különböző feladatokra, így a beszédfelismerésre is. Felderítésük, előnyeik és hátrányaik megismerése és összevetésük az irodalomkutatás során végezhető el.

A neurális hálók természetüknél fogva igen összetettek, ezért gyakori az egyes rendszerek magas erőforrásigénye, melyeket a erőforrásszűkében akár kivitelezhetetlen is lenne használni. Ezért célszerű, hogy olyan architektúrát válasszunk, ami az előzőek alapján optimális teljesítményt nyújt, azaz látványos

és felhasználható eredmények elérésére képes a lehető legrövidebb idő alatt a legkevesebb erőforrást felhasználva. A rövid idő alatt lefutó modellek a tesztelési fázist is nagyban felgyorsíthatják, így az idő szerepe kulcsfontosságú, míg természetesen fontos és nem elhanyagolható a pontosság is.

Kezdetben angol nyelven célszerű tesztelni az megtervezett architektúrát és az egyes paramétereinek pontosítását, mivel angol nyelven rengeteg eredmény érhető el, ezáltal összevethetőek az elért eredmények. Ha egy megtervezett rendszer működésének megítélése angol nyelvű adatok alapján megfelelő akkor már célszerű áttérni az orosz nyelvre.

2.3. Orosz nyelvű implementáció

Egy angoltól eltérő nyelvre való áttéréskor el kell végezni annak a vizsgálatát, hogy az újabb nyelv milyen különbségeket rejt. Egy gyakori eltérés az ábécé, de előfordulhatnák más előfeldolgozásbeli különbségek is. Ha egy keretrendszer angol nyelvre van optimalizálva, előfordulhat, hogy ezeket a finomításokat el kell vetni és egy általános megközelítést kell alkalmazni az új nyelv tanítása során.

A célnyelvi adatokat különböző forrásokból lehet begyűjteni. Mivel az adatbázisok csak nyíltak lehetnek, ezért ezek megbízhatóságát meg kell vizsgálni akár szúrópróbaszerűen akár más felhasználók véleménye alapján. Az adatok mérete mind a neurális modellek tanítási ideje mind tárhelykapacitás miatt nem szabad, hogy túl nagy legyen, meg kell találni az optimális mennyiséget.

Tanítás során a jobb eredmény elérése érdekében alkalmazható a transfer-learning, ami egy elterjedt tanítási módszer neurális hálók használata során. Ez a folyamat egy adott architektúrájú, előre tanított modell továbbtanítását foglalja magába. Beszédfelismerés esetén az eredeti modelltől eltérő nyelv alkalmazásánál kihívást jelent az új nyelvre való áttérés. Szerencsére lehetőség van az előre tanított modell tetszőleges részeit átvenni, nem szükséges a nyelvspecifikus elemeket is importálni, így azok tanítása kezdődhet a többitől szeparálva, nulláról.

Célszerű az architektúra tervezésénél gondolni a szabadon elérhető előre tanított modellekre és ilyen felépítéseket is fontolóra venni és megvizsgálni, hiszen csak ezekkel lehetséges a transfer-learning alkalmazása.

2.4. Optimalizálás és kiértékelés

Az elkészült modellt ki kell értékelni, összehasonlítani más eredményekkel és véleményezni az így elért eredményt. Ezért a kiértékeléshez egy elterjedt módszert kell használni, hogy összemérhetőek legyenek az eredmények.

Lehetőség van az eredmény javítására különböző módszerekkel, melyek már ismertek a neurális hálók körében, vagy főként beszédfelismerésnél jelennek meg. Az előbbi csoportba tartozik például az augmentálás, mely folyamat során a meglévő tanítóhalmazt dúsítják különböző módszerekkel. Utóbbi csoportba tartoznak a különböző dekóderek, annak a módja hogy hogyan értelmezzük a valószínűségként kapott kimeneteket az egyes időpillanatokban, illetve a nyelv modellek.

2.5. Munka összegzése és kitekintés

Befejezettképp kerül sor a kitűzött feladatok sikerességének bemutatására, az elért eredmények részletezéséve és az esetleges kudarcok a zsákutcák leírására. További, ki nem próbált lehetőséget és fejlesztési ötletek javaslása is fontos a munka folytathatósága végett, indokolva, hogy miért is jelenthetnek fejlődést az eredményekben és hogy ezek miért nem lettek megvalósítva, kipróbálva.

3. fejezet

Az end-to-end beszédfelismerés elmélete

3.1. Hagyományostól az end-to-end-ig

A hagyományos ASR-ok (automatikus beszédfelismerők) több elemből álló, összetett rendszerek, melyek manapság igen pontosan meg tudják állapítani mi volt az eredetileg elhangzott szöveg, és leképezik azt írásos formába. Az egyik legelterjedtebb a HMM (Rejtett Markov Modell) alapú beszédfelismerő rendszer. A rendszernek több eleme is van: egy dekóder mely tartalmaz egy nyelv modellt, akusztikus modelleket és egy kiejtési szótárat is, illetve tartalmaz egy elemet, mely a bemeneti hangból a kiemeli jellemzőket, átalakítja őket a rendszer által kezelhető formájúvá. A HMM-en alapuló rendszerek fő eleme a Viterbi algoritmus [16], mely dinamikus programozást alkalmazva találja meg a legjobb illeszkedést a bemeneti szöveg és az adott beszéd modell közt.

A rendszer különböző elemei erősen függenek egymástól. Ha az egyik nem rendeltetés szerűén működik és rossz kimenetet ad, akár saját vagy egy korábbi elem hibájából kifolyólag az befolyásolja a többi elemet is, így elrontva, torzítva a kapott végeredményt. Az egyes elemeket külön-külön kell kalibrálni, tanítani és tökéletesíteni.

Ezzel szemben, a napjainkban egyre inkább népszerű, E2E (end-to-end) alapú beszédfelismerő rendszerek sokkal egyszerűbb struktúrájúak, kevesebb

különálló elemet igénylő megközelítést nyújtanak. Bizonyos szituációkban, ahol egy specifikusabb szövegkörnyezetben, pl. pénzügyi vagy jogi, elhangzó szavak azonosítása a cél, már a hagyományos ASR rendszereknél is pontosabb eredményeket képesek produkálni.

Az E2E rendszerek mély neurális hálókon alapszanak, közvetlen bemenetük a nyers hang, kimenetük pedig a becsült szöveg. Természetesen a neurális háló bemenetéhez a pontosabb eredmény végett a hangot előre fel kell dolgozni, amit pre-processing-nek, előfeldolgozásnak, neveznek. A háló kimenete pedig az egyes bemeneti időegységekre becsült karakterek valószínűségeinek mátrixa melyekből ki kell nyerni a végső, feltételezett szöveget.

3.2. A beszédfelismerés lépései

3.2.1. Előfeldolgozás

A beszéd előfeldolgozása az első fontos lépése a beszédfelismerő rendszereknek. Vannak törekvések, melyekben a E2E neurális hálónak közvetlenül a nyers, feldolgozatlan hanganyagot adják meg és ez alapján tanítják[17], ezáltal még inkább tisztán a neurális háló működésére támaszkodva. Egyelőre ennek a megközelítésnek az eredményei messze elmaradnak a hang előfeldolgozásával elérhető SOTA modellek pontosságától.

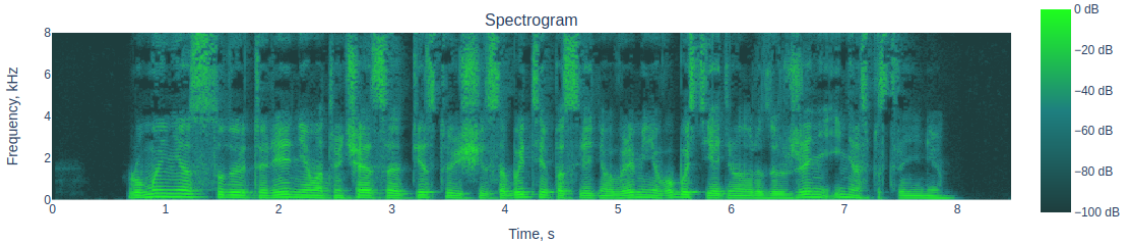
A két fő megközelítés többnyire hasonló, apró eltérésekkel. Az egyik legelterjedtebb és bevált átalakítás a spektogram.

Ahogy az 1. ábrán is látható, a spektogramm három tengely mentén reprezentálja az adatot. A vízszintes tengelyen az idő, míg a függéleges tengelyen a frekvencia van megadva. Van továbbá egy harmadik tengely is, ami színnel van jelölve és mértékegysége a decibel. Ez a jel amplitúdója, ami az adott időpillanatban, adott frekvenciához tartozik és a szín intenzitása reprezentálja a kép pontjain mértékét.

A szokványos spektrogrammon kívül használatos még a MFCCs (Mel-Frequency Cepstral Coefficients) is. Felépítése hasonló a spektogramméval, viszont a frekvencia helyett az MFC koefficiens [13] értéke van az függőleges tengelyen. Ez a

Mel skála, amely úgy van megválasztva, hogy a skálán egyelő távolságra lévő hangok az emberi fül számára is hasonló távolságúnak, azonos léptékben különbözőnek, tűnjenek. A frekvenciához képest egy lényeges változást jelent, hiszen az ember számára könnyen hallható kisebb frekvenciákon, például az 500 Hz és 1000 Hz közötti különbség, míg 6500 Hz és 7000 Hz között már kevésbé számottevő az eltérés. Az MFCC nagyságrendekkel kisebb a frekvencia értéknél így lényegesen befolyásolhatja a neurális háló tanulási folyamatát.

Az előfeldolgozott jelet időegységekre bontjuk, melyek már a neurális háló bemenetét képzik. A kimeneten az egyes időegységekre kapott predikciókból pedig összeállítható a feltételezett szöveg.



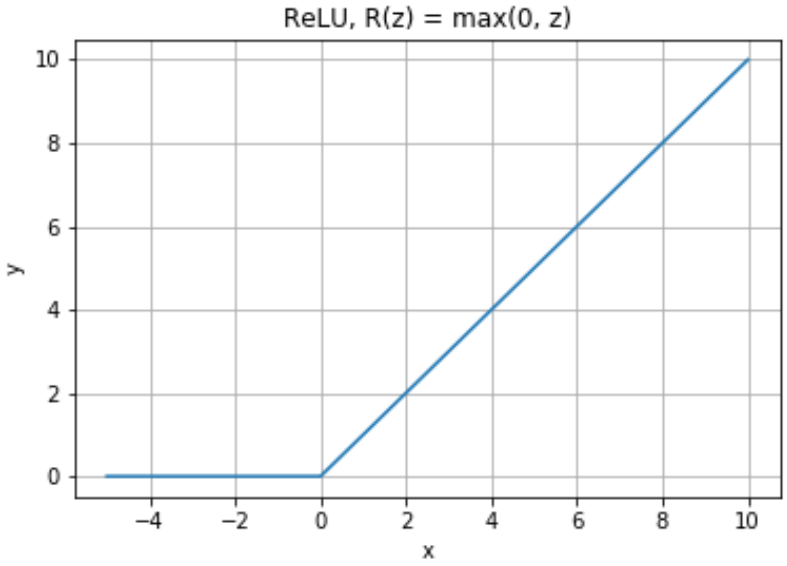
3.1. ábra. Spektogram.

3.2.2. Neurális hálók

A neurális hálók az emberi idegek működésének mintája alapján modellezett láncolatok. Számos különböző, komplex feladat megoldására előszeretettel alkalmazzák őket. Röviden összefoglalva három fő alkotó részből állnak. Egyik a bemenet, például egy kép esetén a pixelek, beszédfelismerés esetén az egyes időpillanatokhoz tartozó értékek vektor formában. Spektogram esetén az időpillanathoz tartozó vektor egyes elemei a frekvenciát jelképezik és az elemek értéke pedig a hangerősséget.

A neurális háló közbülső, rejtettnek, nevezett része tetszőleges méreteket ölthet. Az egyes rétegek pontjai mindig az előző réteg pontjaival vannak összekötve úgy nevezett súlyok segítségével. Az újabb rétegek pontjainak értékét az abba befutó súlyok, illetve a súlyok és kiinduló pontjuk szorzatainak az összegével számítjuk. A pontok végső értéke még egy aktivációs függvénynek nevezett kiértékelésen is keresztül megy, mielőtt a következő réteg értékeit kiszámíttatnánk vele. Az aktivációs

függvények igen eltérőek lehetnek, egy népszerű függvény a ReLU, ami a 3.2-es ábrán látható. Ezeket a pontokat neuronoknak szokás nevezni, innen ered a neurális háló kifejezés. A háló tanítása során ezen súlyok értékét módosítjuk úgy, hogy a legoptimálisabb, számunkra kedvezőnek vélt eredményt kapjuk.



3.2. ábra. A ReLU aktivációs függvény.

Az utolsó réteg a kimeneti réteg, ahol az eredményt kapjuk meg. A kimeneti réteg lehet regresszív, egy előre meg nem határozott, tetszőleges érték, vagy a mint a mi esetünkben klasszifikáció, amikor előre meghatározott címkék (label-ek) valószínűségére vagyunk kíváncsiak. Legegyszerűbb estben a legvalószínűbb címke kerül kiválasztásra válaszként, de a beszédfelismerésnél a később taglalt CTC alapú loss számítás működéséhez az egész kimeneti mátrix vizsgálata szükséges.

A valószínűség kiszámításához a softmax [15] függvényt szokás használni, ami szintén egy aktivációs függvény. A függvény egy vektor bemenetet megkapva normalizálja a vektor egyes értékeit úgy, hogy azok új értékei 0 és 1 közé essenek, miközben az összegük pontosan 1-et adjon. A softmax függvény, ahol x a bemeneti vektort jelöli, míg i és j a vektor egyes elemeinek a sorszámát:

$$softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j))}. \tag{3.1}$$

Neurális hálók egyik leggyakoribb, a munkák során általam is alkalmazott, tanítási módszere a felügyelt tanítás. Ez azt jelenti, hogy először például egy tanítatlan, véletlenszerűen inicializált, hálóból kiindulva kiértékeljük egy tetszőleges bemenetre a végeredményt. Majd a végeredmény helyességétől függően változtatjuk a súlyok értékét a hálóban: azon súlyokat melyek egy rossz eredmény nagyobb valószínűségéhez járultak hozzá büntetjük, úgy csökkentjük az értéküket, hogy kevésbé befolyásolják a végeredményt, míg azokat melyek egy jó eredmény pozitív értékéhez járultnak hozzá növeljük. Ezt a folyamatot szokás backpropagation-nak nevezni. A büntetés mértéke is létfontosságú, hiszen fontos, hogy ne csak az adott bemenetre tudjon pontos végeredményt adni a háló, hanem általánosan is pontos legyen.

Az egyik legegyszerűbb réteg típus a Dense (sűrű) vagy más néven Fully Connected, ahogy a neve is utal rá az egyes neuronok a súlyokkal sűrűn vannak összekötve. Ez azt jelenti, hogy az adott réteg mindegyik neuronja össze van kötve az előző réteg összes neuronjával súlyok által. Több Dense réteg is követheti egymást, mely esetben az adott réteg összes neuronja össze van kötve a következő réteg összes neuronjával is.

Az E2E-nél használt leggyakoribb neurális háló réteg fajták az rekurrens (RNN) és konvolúciós (CNN) rétegek. Ez utóbbit elterjedten használják a különböző képfelismerő feladatok megoldásánál is.

Lényegesen összetettebb a Dense rétegnél a konvolúciós réteg[8], melynél egy, a teljes bemenetnél kisebb, kernel-t (vagy filtert) mozgatunk végig a bemenet elemein és kiértékeljük az általa lefedett értékeket. A lefedett számokat a kernel velük fedésben lévő értékével összeszorozzuk, ezek az értékek főként a -1, 0 és 1, majd a kapott eredményeket összeadjuk. Ezt követően a kernel-t egy megadott értékkel, a stride-al mozgatva kiértékeljük a következő, kernel által lefedett számokat. A konvolúció végezhető egy vagy több dimenziós bemeneten, különböző kernel-ekkel párhuzamosan is, kép esetén például három csatornás képen, ahol a piros kék és zöld értékek külön csatornákon vannak reprezentálva. Egy opcionális lépés, hogy a bement széleit feltöltjük-e nullásokkal, ez az úgy nevezett zero padding. Ennek célja, hogy ugyan akkora legyen a kimenet, mint a padding nélküli bemenet volt. Ezen kívül is szükség lehet padding-re például egy 5x5-ös kép esetén, ha a kernel

mérete 3x3 és a stride 3, akkor ki kell tölteni a kép széleit, hogy minden bemeneti értéken legalább egyszer végig menjen a kernel.

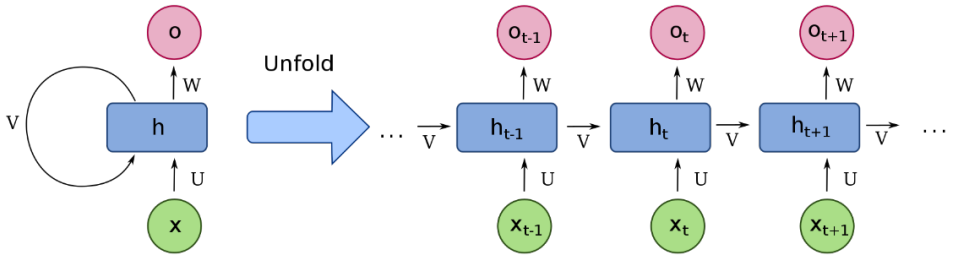
Egy CNN-hez kapcsolódó, kernellel rendelkező rétegtípus a pooling réteg, mely kifejezetten az adat méretének csökkenését szolgálja. Ezzel a célja a számítási kapacitás csökkentése, miközben minél inkább megőrzi az adatban rejlő információt. Két használt pooling típus a max vagy average pooling. Előbbinél a kernel által befoglalt értékek maximumát választja, míg utóbbinál az átlagukat egy számra csökkentve a kernel által közrefogott értékeket.

A konvolúciós vagy pooling rétegek működésükből adódóan csökkenthetik a kimenetük méretét, ezáltal ismerni kell annak pontos dimenzióját, a következő réteghez kapcsolódó számítások végett. A kimenet méretét a következő képlettel lehet kiszámolni, ahol 'O' a kimenet mérete, 'W' a bemenet mérete, 'P' a padding mérete, 'K' a kernel mérete és 'S' a stride:

$$\mathbf{O} = \frac{\mathbf{W} - \mathbf{K} + \mathbf{2P}}{\mathbf{S} + (1)}. \tag{3.2}$$

A rekurrens rétegek[14] egyik legérdekesebb tulajdonsága, hogy nem csupán a bemeneti súlyaikat használják a neuronok kiértékeléséhez, így felhasználva a korábban tanultakat, hanem emlékeznek a korábbi neuron értékeikre is, melyek szintén befolyásolják az aktuális bemenetre kapott eredményt. Beszédfelismerés esetén ezt úgy is el lehet képzelni, hogy ha például elhangzik egy magánhangzó, akkor utána nagy valószínűséggel egy mássalhangzót várunk, ennek fényében, sikeres tanítás esetén, a mássalhangzókat részesíti előnyben egy azt megelőző magánhangzó után. Korábbi időpillanatok, állapotok tehát hozzájárulnak egy újabb állapot kiértékeléséhez a rekurrens rétegben, ahogy ez a 3.3-as ábrán is látható.

Két altípusa a rekurrens hálóknak az LSTM és GRU (Long Short Term Memory, Gated Recurrent Unit). Ezek a típusok hasonló elveken működnek, a különbség a neuronok korábbi értékeinek újabb értékekre történő ráhatásának számítási módjában rejlik. Az LSTM [7] fő célja, hogy kijátssza a hagyományos rekurrens hálók egyik legnagyobb problémáját, azt, hogy a régebbi bemenet súlya nincs hasonló fontossággal kezelve, mint a jelenlegi bemenethez közelebbi érték.



3.3. ábra. A rekurrens réteg ”kigörgetve” [14].

Könnyen belátható, hogy fontos lehet egy mondat végi szót vizsgálva, hogy mi volt a szó a mondat elején, akár a vizsgálandó mondat elemet megelőző néhány szónál is. Az LSTM kapukat használ annak megelőzése céljából, hogy egy korábbi értékelés elvesszen, ezáltal kezelhetetlenné téve annak beleszámítását az újabb értékelésnél.

A GRU egy LSTM-hez hasonló újabb fejlesztésű réteg. Egyik fő tulajdonsága, hogy kevesebb elemet tartalmaz, így kevesebb paramétert használ, csökkentve a neurális háló komplexitását, ezáltal a szükséges számítási időt is.

Megemlítendő még a Bi-directional (két irányú) réteg, amely mögött az a gondolat húzódik meg, hogy ne csak a múltbéli bemenetek befolyásolják az adott bemenetünket, hanem a jövőbeliek is. Két rekurrens réteget használ, melyek ellentétes irányban haladnak egymással, így tetszőleges kiértékelésnél használhatóak a korábbi, illetve elkövetkezendő neuronok eredményei is.

A rekurrens rétegek nagy léptékben meg tudják emelni a modellek paraméterszámát, ezért használatuktól igyekeztem tartózkodni, mivel az architektúra kiválasztásának fő szempontjai az alacsony paraméterszám, és a gyors kiértékelési és tanítási sebesség voltak.

3.2.3. CTC dekódolás és loss számítás

A CTC (Connectionist Temporal Classification) [6] funkciója a kimenet dekódolása, a válasz kiértékelése, mely folyamán átalakítja a kimenetet a megadott címkék valószínűségi szekvenciájára és a tanításhoz szükséges hiba, a loss kiszámítása. A címkék lehetnek az egyes karakterek, amik egy adott nyelvben előfordulhatnak. Ez esetben a CTC kiértékeli, hogy melyik betű vagy egyéb karakter hangzott el

adott időpillanatban, adott bemenetre [8]. Ereje abban rejlik, hogy nincs szükség a kiértékelésnél a kimeneteket egyenként, karakter szinten összekötni az elvárt szöveggel, elegendő a kimeneti szöveget megadni, azt, amit kapni szeretnék az adott bemenetre. Másik fontos előnye, hogy a felismert szöveget nem szükséges külön feldolgozni, mivel átalakítja azt a végleges, feltételezett formára. Feldolgozza, hogy egy hosszú hang esetén, mint az 'ú' több időpillanaton keresztül is hallatszik az 'ú' betű, de nekünk a végeredményben csak egy darab 'ú' szükséges.

Felmerülhet egy probléma az összevonással, amikor olyan egymást követő betűket vonunk össze, melyek a szóban külön szerepelnek, mint például az 'ellentét' szóban. Ennek elkerülése végett bevezet a CTC egy üres karaktert, ami nem egyenlő a szóközzel, és ezt olyan időegységekhez helyezi, ahol nem ismerhető fel betű a megadott szótárból, karaktergyűjteményből. Az ilyen üres karakterrel elválasztott betűket nem fogja összevonni kiértékeléskor.

A neurális háló által végzett kiértékelés után az ember által feldolgozható kimenet megszerzéséig szükség van még a karakterek összeillesztésére. Ehhez különböző elven működő dekódolókat használunk. Az egyik legalapvetőbb dekódolási algoritmus a best path eljárás. Két dolgot végez: kiszámolja a legvalószínűbb útvonalat a kapott kimeneti időegységeken át, majd eltávolítja az egymást követő azonos karaktereket, melyek közt nem található az üres karakter. Végül az üres karaktereket is eltávolítja, így eredményezve a végső predikciót.

További feladata a neurális háló tanításához használt loss kiszámítása [10]. A hiba számítása során a szempontunkból lényegtelen milyen karaktersorozattal jut el a végeredményhez, csupán az a cél, hogy pontosan az adott kimenetet eredményezze. A loss-t tehát úgy számítja, hogy megnézi az összes olyan karaktersorozat valószínűségét, melyből megkaphatja a kívánt eredményt és összegzi őket. Ezáltal az eredeti GT (Ground Truth) szöveg valószínűségi értéke magas, hogyha a megszerzéséhez szükséges kombinációk értékének az összege magas.

A tanításhoz szükséges CTC loss számítása a 3.3-as, 3.4-es és 3.5-ös képeltek alapján történik, ahol GT az elvárt szöveg, X az egyes akusztikus keretek, C egy adott karaktersorozat és c_t egy adott karakter adott időpillanatban.

$$CTCloss = -\log P(GT|X). \tag{3.3}$$

Összes lehetséges helyes karaktersorozat valószínűségének az összege:

$$P(GT|X) = \sum_{C=A(GT)} P(C|X). \tag{3.4}$$

Egy adott, helyes karaktersorozat valószínűsége:

$$P(C|X) = \prod_{t=1}^{T} y(c_t, t)). \tag{3.5}$$

A neurális háló kimenetén megadja a megadott karakterek valószínűségi mátrixát minden időpillanatban. Mivel a loss kiszámítása a GT szöveg összes lehetséges illeszkedés alapján történik, ezért lényegtelen, hogy hogyan helyezkedik el pontosan a megadott hanganyagban. A fenti képletekkel kiszámítható egy tanító szöveg valószínűsége a megadott GT és a neurális háló által adott valószínűségi mátrix alapján. A cél, hogy aképpen tanítsuk a hálót, hogy magas valószínűségeket kapjunk az elvárt ground truth kimenetkre. Ezt úgy lehet elérni, hogy minimalizáljuk a tanítóadathalmazra kapott loss értéket, ahol a loss a valószínűségek logaritmikus értéknek negatív összege. Az egyes loss értékek kiszámításához a valószínűség kiértékelése után elég annak logaritmusát venni, majd negálni az eredményt. A neuron háló tanításához a loss grádiensét kell kiszámolni, azt az értéket amivel a többdimenziós térben a legnagyobb csökkenést lehet elérni, és az egyes paramétereket, súlyokat frissíteni vele[6].

3.3. Létező megoldások

A beszédfelismerő rendszereket pontosságuk alapján tudjuk rangsorolni. Egy nemzetközileg elismert pontossági mérőszám a WER (Word Error Rate). Ezt a pontosságot szokás szerint egy publikusan elérhető, angol nyelvű hangoskönyv felolvasásokat tartalmazó adatbázison, a LibriSpeech-en[1] mérik.

Különböző, alacsony hibaarányú modellek léteznek, de ez egyik lekiemelkedőbb közülök az Nvidia QuartzNet modellje, mely alacsony számú paraméterek mellett képes igen nagy precíziót fenntartani. Míg a QuartzNet bonyolultsága,

[1]LibriSpeech oldala: `http://www.openslr.org/12/`

paramétereinek a száma csupán töredéke a legtöbb SOTA modellének[2], a WER száma megközelíti azokét (3.1-es táblázat).

A modellekben elterjedt a konvolúciós- (CNN) és rekurrens neurális rétegek (RNN) használata.

Model	**Aug**	**LM**	**clean WER(%)**	**other WER(%)**	**Million Parameters**
wav2letter++[2]	Speed Perturb	ConvLM	3.26	10.47	208
LAS[1]	SpecAugment	RNN	2.5	5.8	360
Time-Depth Separable Convolutions[5]	Dropout, Label Smoothing	N/A (greedy)	5.36	15.64	37
		4-gram	4.21	11.87	
		ConvLM	3.28	9.84	
Multi-Stream Self-Attention[4]	Speed Perturb	4-gram	2.93	8.32	23
		4-LSTM	2.20	5.82	
QuartzNet-15x5[9]	Spec Cutout, Speed Perturb	N/A (greedy)	3.90	11.28	19
		6-gram	2.96	8.07	
		Transformer-XL	2.69	7.25	

3.1. táblázat. Az egyes SOTA beszédfelismerő modellek és paramétereik.

[2]Az Nvidia ismertető oldala: `https://developer.nvidia.com/blog/develop-smaller-speech-recognition-models-with-nvidias-nemo-framework/`

4. fejezet

Tervezés

4.1. Tervezés, döntési lehetőségek értékelése

4.1.1. Fejlesztési nyelv

A Python napjaink egyik legelterjedtebb programozási nyelve. Egy könnyen programozható és átlátható, interpretált nyelv, melyben a megírt program kódot a Python értelmező sorról-sorra értelmezi és futtatja, nincs különválasztva a forrás és a fordított kód. Deep learning körökben is dominál a Python nyelv, rengeteg deep learning toolkit erre a nyelvre épít, így választásom a fejlesztéshez természetesen a Python nyelvre esett.

4.1.2. NeMo

A neurális hálók komplikáltságából adódik, hogy pontos és precíz implementációjuk igen bonyolult és időigényes. Emiatt célszerű a már meglévő, bejáratott és bizonyított lehetőségeket használni, így felgyorsítva a fejlesztési folyamatot és javítva a potenciálisan elért eredményeket. Több deep learing platform is található az internetet, mint például a Tensorflow vagy PyTorch. Az Nvidia által fejlesztett NeMo[1] egy olyan összetett és folyamatosan karbantartott toolkit, melynek segítségével különböző mesterséges intelligencia alapú társalgási, beszéddel kapcsolatos applikációkat, alkalmazásokat készíthetünk. Munkám során ennek fényében a NeMo toolkit-et használtam.

[1]NeMo toolkit, `https://github.com/NVIDIA/NeMo`

Az alap koncepcióját a toolkit-nek az úgynevezett Neurális Modulok alkotják. Ezek a modulok lehetnek például adat rétegek, kódolók, dekóderek, nyelv modellek vagy loss függvények. A modulok ereje abban rejlik hogy egymáshoz viszonyítva tetszőlegesen építhetők, cserélhetők, törölhetők. Egy egyszerű módosítással kicserélhető például a loss számításához használt függvény anélkül, hogy a kódot a többi modulban módosítani kéne.

Az 3.1-es táblán látható modellek közül az Nvidia fejlesztette a legkevésbé számításigényeset, mely figyelemreméltó eredményeket produkál. Ez a QuartzNet modell, ami az alacsony paraméterszáma ellenére SOTA eredményeket ér el. A felsorolt okokból kifolyólag a fejlesztési folyamat elvégzéséhez választásom a NeMo toolkit-re esett, melyenek 1.0.0b verziójával dolgoztam.

4.1.2.1. PyTorch

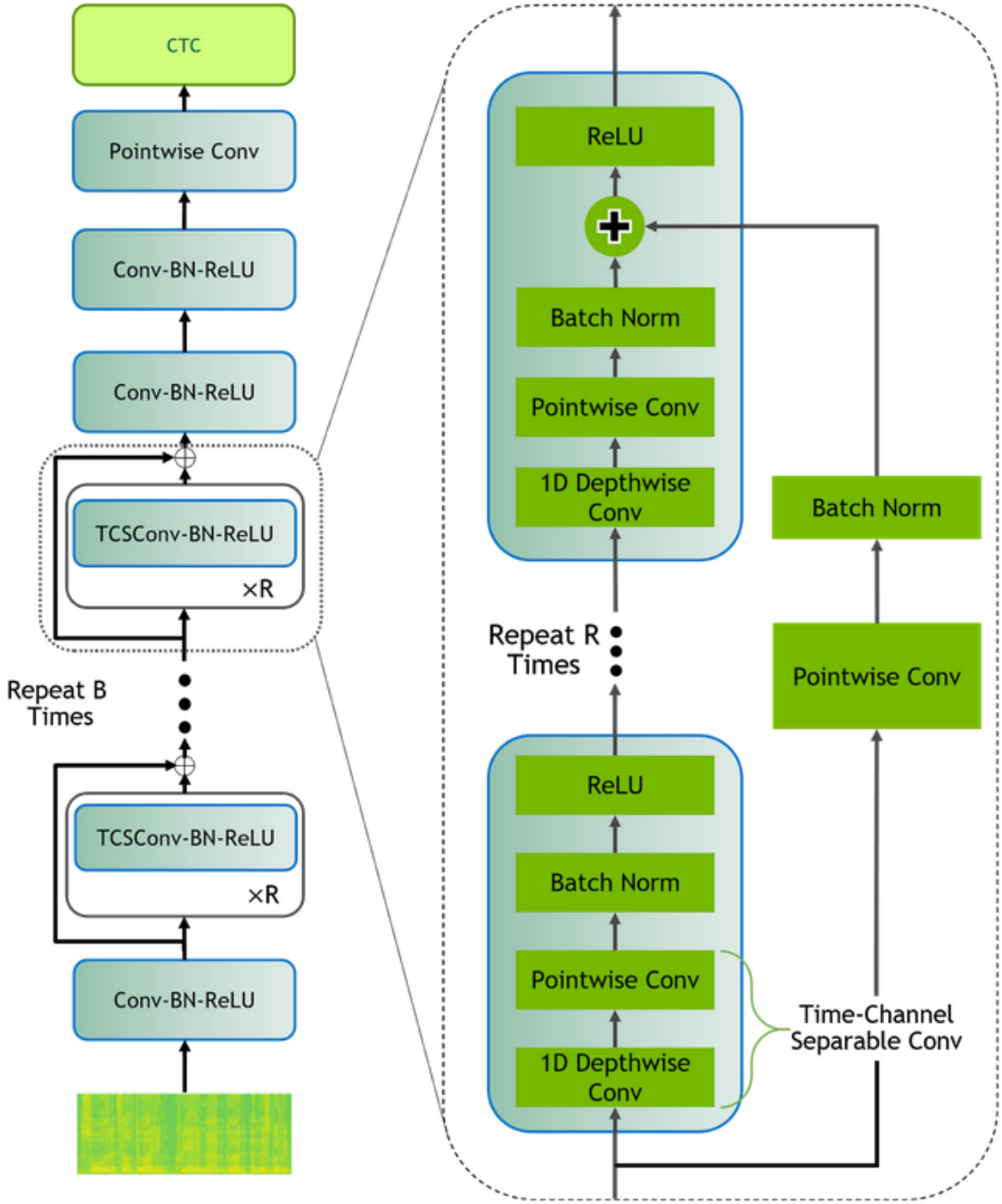
A NeMo a PyTorch nevű, Facebook-os kutatók által fejlesztett, mesterséges intelligencia fejlesztésére szakosodott nyílt forráskódú keretrendszerre épít. A PyTorch a TensorFlow-nál magasabb szintű, könnyebben kezelhető keretrendszer, mely egyre inkább elterjedtté válik a kutatók és fejlesztők körében. A NeMo működéséhez 1.6+ verzió szükséges, én az 1.6-os verziót alkalmaztam.

4.1.2.2. QuartzNet

A kísérleteimet QuartzNet architektúrájú modellekkel elvégeztem, a korábban említett alacsony paraméterszáma és magas precizitása végett. Az architektúra (4.1-es ábra) nagyban hasonlít egy másik Nvidia által fejlesztett architektúrához, a Jasper-hez. A modell B darab blokkból áll. Egy opcionálisan beállítható dropout modul is megjelenhet a modellben, illetve CTC loss függvény számítás található benne. A már említett blokkok további, R darab blokkból állnak, melyek konvolúciós rétegekből, batch normalizálókból és ReLU aktivációs függvényből épülnek fel. Ezáltal a QuartzNet pontos felépítésére QuartzNet BxR alakban szokás hivatkozni. A legnagyobb Nvidia által készített a Quartznet 15x5, mely 18.9 millió paraméterből áll, amit kiterjedten használtam a kísérleteim során.

A QuartzNet a Jasper-től a konvolúciós rétegeinek a típusában tér el. A sima egy dimenziós konvolúciós réteg helyett, egy dimenziós time-channel separable

konvolúciós rétegeket használ [9]. Ennek a fő tulajdonsága, hogy jóval kevesebb paramétert használ az eredeti koncepciónál. Ennek köszönhetően mélyebb hálókat lehet készíteni nagyobb konvolúciós kernelekkel anélkül, hogy a a paraméterek száma, ezáltal a szükséges számítása kapacitás túlzottan megnövekedne.



4.1. ábra. QuartzNet architektúra. [7]

4.1.3. TensorBoard

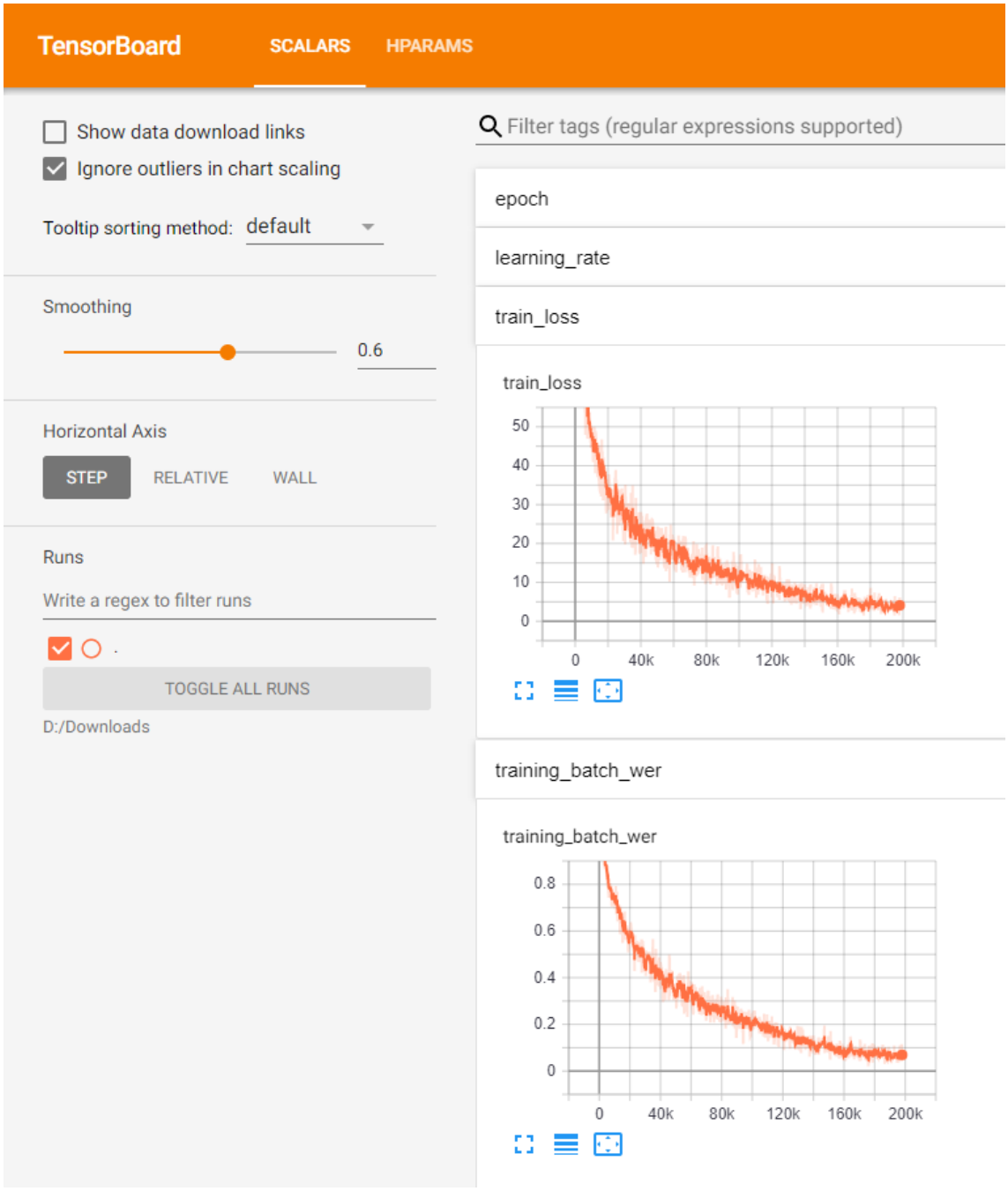
Az eredmények kiértékeléséhez és nyomon követéséhez a TensorBoard-ot, a TensorFlow egy vizualizációs toolkit-jét használtam. Több funkcióval is rendelkezik, mint például a súlyok időbeni változásának vizualizációja, illetve a tanítóadatok, képek, szöveg vagy hang anyag kijelzése. Számomra a legfontosabb tulajdonsága az egyes metrikák, a loss és a pontosság, a WER, értékének változásának nyomon követése.

A TensorBoard könnyedén indítható a következő parancs kiadásával: $tensorboard - -logdir\ /lightning - logs$, ahol a logdir kapcsoló adja meg melyik könyvtárban keressen TensorBoard kompatibilis log-okat, event-eket. A főkönyvtárban egyes almappáiban kell elhelyezni a tanítás során előállított event-eket, így a Tensorboard különálló tanításként ismeri fel őket. A szükséges event-ek előállítása könnyedén végezhető a NeMo toolkit segítségével, mely támogatja a megfelelő callback függvények használatát, amik az event-ek batch-enkénti frissítsét tesznek lehetővé. A callback függvény egyszerűen bővíthető szükség szerint.

4.2. Angol, majd orosz

Kezdetben a tesztelést az egyik leggyakoribb nyelven, az angolon, végeztem. Ennek oka főként az volt, hogy megbizonyosodjak a modellem és a NeMo beállításainak sikerességéről, hogy megfelelő eredményeket tudok elérni elterjedt, kisebb adatbázisokon. Másfelől a modellem hiperparamétereit, mint például a learning rate, optimalizáltam, és az orosz nyelven történő kísérletezésnek a legbiztatóbb beállításokkal tudtam nekiállni.

Az angol nyelvű eredményeket viszonyításként is fel tudtam használni az végső orosz eredményekkel való összehasonlításánál, ahol a nagyságrendileg azonos mennyiségű orosz adatoknak az angoléhoz hasonló eredményeket kellett produkálniuk.



4.2. ábra. Lokálisan futtatott TensorBoard.

4.2.1. Google Colab

A NeMo-val való ismerkedéshez felhasználtam egy, az Nvidia által írt, jupyter notebook-ot[2], melyet könnyedén és ingyenesen lehet Google Colab-ból[3] futtatni. A Colab ingyenes hozzáférést nyújt GPU erőforrásokhoz, ahol egyenesen a böngészőből

[2]Tutorial notebook, `https://github.com/NVIDIA/NeMo/blob/main/tutorials/asr/01_ASR_with_NeMo.ipynb`

[3]Google Colab, `https://colab.research.google.com/notebooks/intro.ipynb`

futtatható a Python kód és az eredményeket akár le is lehet tölteni. Csupán be kell importálni a Colab környezetbe a github-on elérhető notebook-ot a az URL megadásával, majd tömbönként futtatható a kód, melyhez magyarázat is tartozik.

Sajnos a Colab bizonyos órán belül felfüggeszti a munkamenetet, illetve a nagy mennyiségű adatok munkamenetenkénti le- és feltöltése időigényes. Ezért főleg a kisebb, rövidebb adatbázisokon, mint például az AN4-en, lehetséges a kísérletezés és a toolkit-tel való ismerkedésen túl nem jelent megoldást.

4.3. Adatbázisok

A megfelelő modell megtalálásához két darab angol nyelvű adatbázist is használtam. Mikor elégedett voltam az eredményekkel és megbizonyosodtam, hogy sikeresen beállítottam a NeMo toolkit-et is, áttértem az orosz nyelvű modell fejlesztésére.

Az adatbázisokon belül három különböző típusú, egymástól független adathalmazt szükséges megállapítani. Az egyik a tanító adathalmaz, ezeket a hanganyagokat közvetlen a háló tanítására használjuk. Fontos, hogy legyen egy validációs vagy teszt adathalmaz is, amit tanítás közben használunk kiértékelésre, a tanítás általános pontosságának értékelésére. A harmadik dev adathalmaz pedig azért szükséges, mert a validációs adathalmaz pontosságát igyekszünk javítani, majd amikor ezzel elégedettek vagyunk egy tőle független adathalmazzal vizsgálhatjuk meg a végső pontosságot.

4.3.1. Adatok előfeldolgozása

Magukat a hangfájlokat is meg kell vizsgálni, egységesíteni kell a tanítás előtt. A leggyakrabban használt kiterjesztés a .wav, így én is ennek a használata mellett döntöttem. Szükség esetén az egyes hangfájlokat át kell konvertálni ebbe a formátumba a tanítás, kiértékelés előtt.

A hanganyagok megfelelő mintavételi frekvenciájának megválasztása fontos feladat. A túl alacsony mintavételezés minőségi romlást, rosszabb eredményeket okozhat, mivel a magasabb frekvenciakomponensek eldobásra kerülnek. A túl magas mintavételezés lassítja a tanítási folyamatot és akár szintén rosszabb eredményeket produkálhat.

Az adatbázisok esetén megvizsgálandó, hogy hány beszélő hangja található benne. Minél több van benne annál általánosabban alkalmazható jobb eredményekkel.

4.3.2. AN4

Az AN4[3] vagy ”census” angol nyelvű beszéd adatbázist 1991-ben vették fel a Carnegie Mellon Egyetemen. A különböző beszélők betűzve mondanak olyan adatokat mint a születési dátumuk, telefonszámuk, a nevük vagy egyéb véletlenszerűen generált, előre definiált kontroll szavak.

Az adatbázis két különböző részre van osztva, az első a tanítást szolgálja, míg a második része a tesztelést. Ez előbbi 50 percnyi beszédet tartalmaz, míg a tesztelésre szolgáló rész 6 percet. Rövidségéből adódóan gyorsan tanítható és értékelhető, mely ideálissá teszi gyors kalibrálási, beállítási feladatok megoldására.

4.3.3. LibriSpeech

A LibriSpeech egy több száz órát tartalmazó, szintén angol nyelű, hangoskönyv gyűjtemény. Ez az egyik leginkább használt adatbázis, melyet előszeretettel alkalmaznak tudományos, kutatási munkák során, hálók tanításához és ezáltal az eredményeinek ismertetésekor.

Az elérhető hanganyag fele tiszta, könnyen érthető, míg másik fele zajosabb körülmények közt lett felvéve vagy kevésbé kivehető. Külön tartalmaz tanító, teszt és dev adathalmazokat. A tanításhoz a 100 órányi tiszta adathalmazt, a train-clean-100-at alkalmaztam.

4.3.4. Mozilla Common Voice

A Mozilla Common Voice[4] egy Mozilla által indított projekt. Célja, hogy különböző nyelveken, így oroszul is, a közösség erejét felhasználva gyűjtsön anonim hanganyagokat. Bárki felveheti a saját hangját, előre meghatározott szövegeket felolvasva, illetve érvényesíthet mások által felolvasott szövegeket. A biztonság és pontosság kedvéért egy szöveget két személynek is jóvá kell hagynia.

[4]Mozilla Common Voice weboldala, `https://voice.mozilla.org/`

Orosz hangból durván 100 órányi áll rendelkezésre, de tüzetesebb vizsgálat után észrevettem, hogy ebből 10-10 óra a teszt és dev adathalmaz mérete, míg 20 óra a tanító adathalmazé. A maradék 60 óra nagy része azért nincs használva, mert megismételt szöveget olvasnak fel benne, vagy rövid hanganyagokat tartalmaz. A tanító adathalmazt sikerült feldúsítsam 45 óra környékére azáltal, hogy belevettem a 100 órányi hangból az összes olyan hangot, melyek nem voltak benne sem a teszt sem a dev adathalmazokban.

4.3.5. További orosz adatbázisok

Mivel a Mozilla Common Voice orosz nyelvű adatbázisa legfeljebb 45 órányi tanítóadattal rendelkezik, míg a LibriSpeech-é mely eredményeihez viszonyítani szeretnénk 100 órán lett tanítva, ezért több tanítóadatot is be kellett vonni. Egy ingyenesen hozzáférhető github-on található projektben[5] több adatbázis is található, többnyire .opus formátumban. Ezek közül elsősorban a "Radio2" adatbázissal dolgoztam.

Fontos megemlíteni a Mozzila Common Voice és egyéb rádiós műsorban elhangzott hanganyagok közötti lehetséges különbségeket. Míg előbbi többnyire bediktált, felolvasott szöveget tartalmaz különböző minőségben és tempóban, utóbbi kötetlenebb, tisztább minőségű beszélgetéseket is tartalmazhat.

4.4. A tervezés eredményei

Megoldásomban az Nvidia QuartzNet alapján indultam el, annak alacsony paraméterszámát figyelembe véve. Egy Pytorch alapú toolkit-et, az Nvidia NeMo-t (Neural Modules) használva kísérleteztem QuartzNet kombinációkkal.

Főként az ingyenesen hozzáférhető Mozilla Common Voice adatbázist használva tanítottam orosz nyelvre két architektúrájú modellt a 15x5 és a 12x1-et. A modellt véletlenszerűen beállított súlyokkal és transfer learning-et alkalmazva, egy előre tanított angol modellbeli súlyokkal inicializáltam. Az eredmény finomítása végett

[5]Ingyenes orosz adatbázisok, `https://github.com/snakers4/open_stt/blob/master/README.md`

próbálkoztam több tanítóadat beiktatásával, mellyel a validációs adathalmaz WER értéke csökkenése, annak javítása volt a célom.

A 4.1-es táblázatban láthatóak a felhasznált adatbázisok, azok származási helye, fontosabb paraméterei és elnevezése a diplomamunkán belül.

Elnevezés	**Adatbázis Neve**	**Adatok hossza órában**	**Nyelv**	**Minőség**
an4	AN4	1	angol	tiszta
LS100	LibriSpeech	100	angol	tiszta
orosz_rövid	Mozilla Common Voice	19	orosz	változó
orosz_közepes	Mozilla Common Voice	45	orosz	változó
orosz_hosszú	Radio2	194	orosz	tiszta

4.1. táblázat. Felhasznált adatbázisok ismertetése és elnevezése.

5. fejezet

Implementáció

5.1. Tanítóadatok előkészítése

Az adatok kiválasztása utáni első lépésként a letöltött tanítóadatokat NeMo által értelmezhető formátumúra kellett hozni. Ez magába foglalta a fájlok konvertálását .wav formátumba. Az egyes tanítóadatok más-más formátumban (.opus, .flac, .mp3) voltak eredetileg, illetve eltérő fájlstruktúrákba voltak szervezve.

A konvertáláshoz a fentiek miatt a python os könyvtárának walk metódusát használtam, mellyel egy mappát rekurzívan be lehet járni. A konvertáláshoz az ffmpeg szoftvert használtam, mivel segítségével több formátumból is könnyedén lehet .wav formátumba konvertálni.

A fájlok átalakításán kívül szükséges volt az adatok .json fájljainak elkészítése, mely megadja a programnak, hogy melyik hangfájl hol található, milyen hosszú és mi hangzik el rajta. Természetesen több információ tárolását is támogatja a NeMo, de ezek az adatok elengedhetetlenek. A NeMo-nak ezután csupán az egyes manifesteket, az elkészült .json fájlokat, kellett megadni, azzal a kiegészítéssel, hogy melyiket kívánjuk tanító vagy teszt adathalmaznak használni, és a további előfeldolgozási lépéseket és betöltéseket már a toolkit automatikusan elvégezte.

5.2. Modellek kimentése és betöltése

A tanított modelleket el kell menteni későbbi használat végett. A modellek mentéséhez két módszert is alkalmaztam. Az egyik a logoláshoz hasonlóan callback-

et használ, mindig a legoptimálisabb eredményt elért epochot mentette ki .ckpt (checkpoint) formátumban, míg az utóbbi módszer a tanítás végén mentette ki a modellt .nemo formátumban, ehhez egy a toolkit által megvalósított trainer elem függvényét használtam.

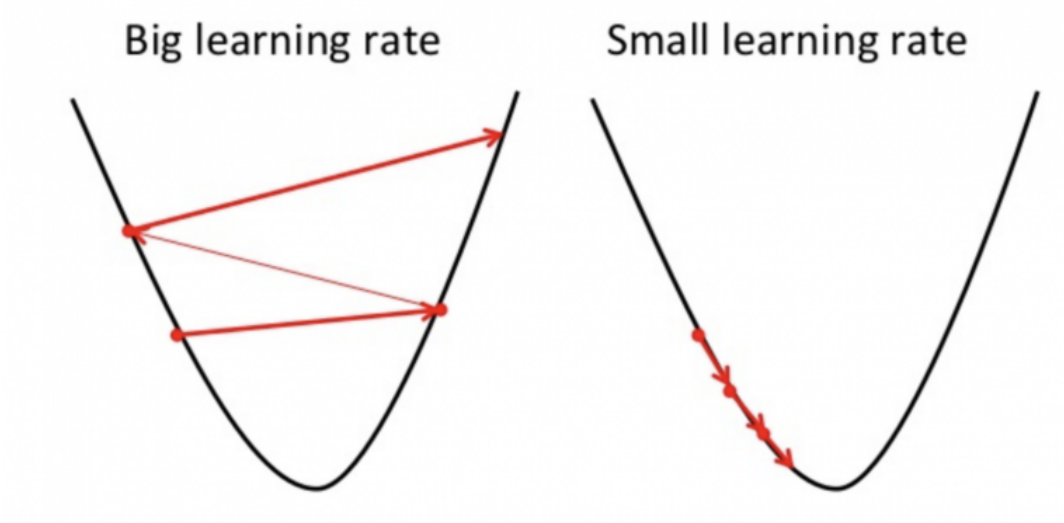
A mentésre a későbbi felhasználás mellett azért is van szükség, mert a hosszú tanításokat, melyek akár 48 órán át is tarthatnak, szükséges lehet megszakítani. A megszakítás után fontos úgy betölteni a modellt, hogy az továbbtanítható legyen. A NeMo külön kezeli a modellt és a tanítást végző trainer-t, melyekbe külön-külön szükséges betölteni a folytatni kívánt checkpoint-ot.

5.3. Kalibráció angol nyelv segítségével

Első lépésként egy kisebb adatbázison, az an4-en teszteltem a NeMo keretrendszert, a Google Colab segítségével. Az alap paraméterekkel és konfigurációval az elért eredmények elmaradtak a referencia eredményekéhez képest. Az an4-es tanítás során QuartzNet 5x1-es architektúrájával dolgoztam a kis mennyiségű adat és gyorsabb tanítás végett.

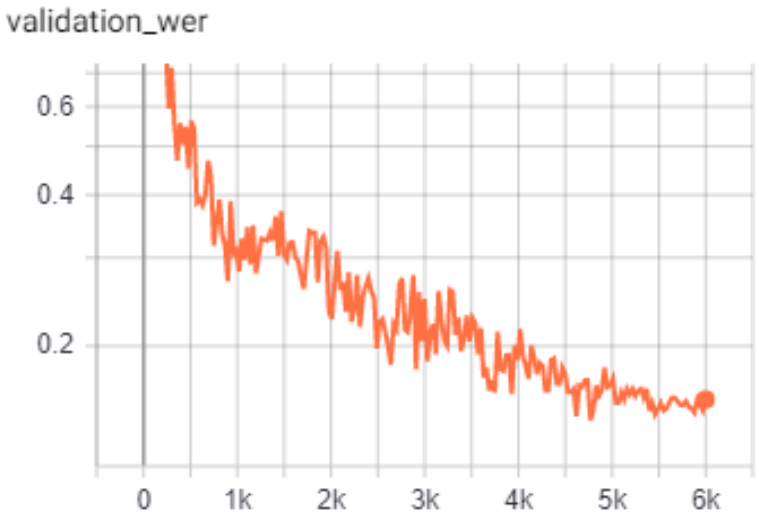
A pontosság növelése érdekében az egy gyakran használt módszer az epoch szám növelése. Egy epoch reprezentálja az összes tanítóadaton való kiértékelést és a súlyok javítását a hiba tükrében. Magas epoch szám jobb eredményeket produkál, de vigyázni kell, mert túlzott epoch szám esetében megjelenhet az úgy nevezett túltanítás jelenség. Ez annyit tesz, hogy a túlzottan is megtanulja a modell a tanítóadatokat, azokra nagyon jó eredményt ad, míg általánosan egyre rosszabbul teljesít.

Egy másik fontos paraméter aminek a módosításával növelni lehet az elért pontosságot a learning rate. A learning rate felel a hiba nyomán felmerült tanítás, azaz súlyok korrekciójának mértékéért. Túl nagy lépések, javítások, esetén az új loss érték, amit csökkenteni szeretnénk, túl nagy ugrásokat végez, potenciálisan a rossz irányba. Túl alacsony érték esetén lassan éri el a loss a minimum pontját, legyen az globális vagy lokális, ahogy az az 5.1-es ábrán látható. Szintén nem kívánt eredmény az alacsony learning rate esetén, hogy a loss értéke ideje korán beragad egy lokális minimum értékben, ami egy nagyobb lépéssel áthidalható lenne.



5.1. ábra. Learning Rate fontossága, `https://builtin.com/data-science/gradient-descent`.

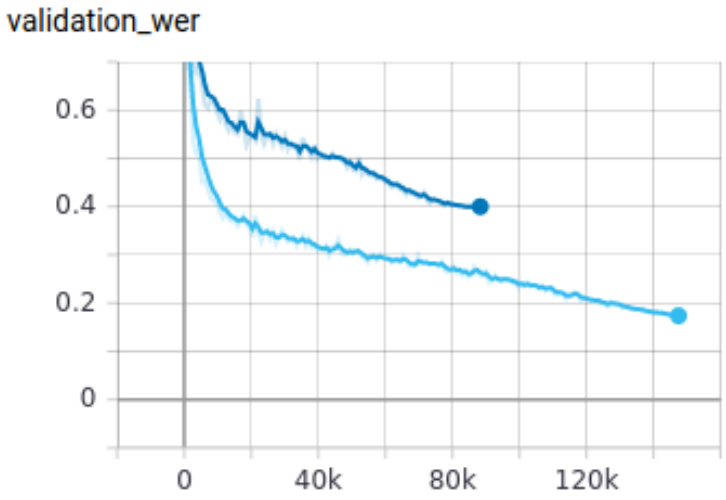
A fentiek tükrében kis kísérletezés után enyhén növeltem a kezdeti epoch számot és a learning rate-et, rendre 200-ra és 0.01-re. Sikerült egy optimálisabb eredményt elérjek, így a felhasznált paraméterekkel tovább tudtam indulni egy nagyobb adatbázison való tesztelésre.



5.2. ábra. AN4 adatbázis validációs adathalmazán elért eredmények.

Arról, hogy a beállításaim tényleg megfelelőek a LS100 adathalmazon bizonyosodtam meg. Azért nem ezzel kezdtem a kalibrálást, mert 100 órányi hanganyag tanítása jóval tovább tart, így minél kevesebbszer volt célszerű végigfuttatni a tanítást az optimalizálás gyorsasága érdekében.

A LibriSpeech-hez egy nagyobb méretű architektúrát használtam. A mélyebb hálók jellemzően pontosabb eredményeket képesek elérni a paraméterszám növelése mellett. Összevetettem az eredeti, 5x1-es modellt a nagyobb, 12x1-es QuartzNet modellel és a korábban optimálisnak vélt epoch számmal és egy magasabb learning rate-el. A különbség az 5.2-es ábrán látható. Látható, hogy a nagyobb, mélyebb hálóval pontosabb eredmények voltak elérhetőek.



5.3. ábra. LibriSpeech eredmények QuartzNet 5x1 (sötétkék) és 12x1 (világoskék) architektúrákon. Merőleges tengelyen a lépések, batchek száma, függőleges tengelyen a WER értéke.

5.4. Orosz nyelvű hálók

Az orosz nyelvű implementációhoz jó kiindulópontot jelentett a LS100-nál is alkalmazott architektúra. Az Mozilla Common Voice orosz adatbázisa átesett a már említett szükséges formázásokon, és legeneráltam belőle a megfelelő .json fájlokat. A generált fájlokban odafigyeltem, hogy csak cirill betűk szerepeljenek, illetve szükség esetén kisbetűsítettem a szöveget.

5.4.1. NeMo config fájl

A NeMo a modellek leírásához config fájlokat használ. A fájlokban minden modellt és tanítást leíró paramétert meg lehet adni. Ezek közé tartozik a modell architektúrája,

a használandó aktivációs rétegek, a learning rate és az epoch szám vagy az optimalizáló típusa.

Előnye, hogy könnyen módosítható és a programkódtól független. Megadhatók üresen hagyott paraméterek is, melyeket ???-el lehet jelölni, vagy egyszerűen nem szükséges őket előre, elég csak futtatáskor a forráskódban definiálni, megadni. Ezeket később programkódból ki lehet tölteni, mint például a használandó manifest fájl elérési útját.

Több config fájlt is használtam, hogy azokat egyszerűen cserélgetve tudjak az előre definiált architektúrák közt váltani.

5.4.2. Szükséges módosítások

A config fájlban két dolgot kellett átállítani az orosz nyelv használata előtt. Az egyik maguk a betűk, label-ek voltak, mivel az orosz ábécé más betűket használ, eltér az angoltól. A cirill karakterek könnyen kinyerhetőek voltak a google translate orosz nyelvű virtuális billentyűzete segítségével.

A másik átállítandó paraméter a config fájl AudioToTextDataLayer részében a leiratok normalizálására szolgáló rész[1]. Erre azért van szükség, mert alapból elvégezné a modell a toolkit beállítási alapján, de szeretnénk elkerülni, mivel angol nyelvre lett tervezve így eltérő nyelveken nem működne megfelelően.

5.4.3. Architektúrák

A kísérletezéshez 2 különböző architektúrát, a már említett QuartzNet12x1-t és 15x5-et használtam. A választáson fő szempontja az volt, hogy ilyen architektúrákhoz rendelkeztem előre tanított angol nyelvű modellekkel. Ezeket később a transfer learning-nél használtam fel, és összehasonlítottam a véletlenszerűen inicializált súlyú modellek eredményeivel.

A tanításokat a Mozilla és Radio2 adatbázisán is futtattam. A kezdeti orosz_hosszú adathalmaz túl nagynak bizonyult, a hanganyagok méretét a config-ban 8.7 másodpercben maximalizáltam, szemben a megszokott 16.7 másodperccel, így az összes tanítóadat 193.75 órát tett ki.

[1] `https://github.com/NVIDIA/NeMo/issues/234`

Fontos kiemelni, hogy a tanítás közbeni pontosság teszteléséhez, azaz a validációhoz mindegyik esetben a Mozilla adatbázishoz tartozó tesztadatokat használtam. A pontosságot külön megvizsgáltam a Radio2-es adatbázisból leválasztott, tanításhoz fel nem használt, tehát 8.7 másodpercnél hosszabb, hanganyagokon is.

5.4.4. Transfer learning

A Transfer learning-hez előre betanított modelleket kellett szerezzek. A QuartzNet 15x5 architektúrájú modell könnyen elérhető volt, az Nvidia által szabadon rendelkezésre volt bocsájtva. A 12x1-es architektúrát konzulensem Dr. Mihajlik Péter bocsájtott rendelkezésemre. Ez a modell kevesebb adaton és rövidebb ideig tanult mint a 15x5-ös, de sokkal jobb kiindulópont volt, aminek előállítására lehetőségem lett volna.

A modell betöltése hasonlóan történt a checkpoint-ból való folytatás betöltéséhez, azzal a különbséggel, hogy nem a teljese modellt használtam, csupán az encoder részét. Ez fontos különbség, hiszen a decoder használhatatlan lett volna, mivel az angol nyelvre lett tanítva. Így a decoder-hez véletlenszerűen inicializált súlyokat használtam, és a modell encoder részének súlyai kerültek átemelésre.

A tanítás ezen felül a megszokottak szerint zajlott a megfelelő config fájl betöltésével, tanítóadatok és trainer beállításával, a súlyok kimentésével. A transfer learning-el tanított modellek új, a Tensorboard által betölthető, event fájlokat hoztak létre.

6. fejezet

Összefoglalás

Architektúra	Adathalmaz	Transfer Learning (I/N)	Tanítás hossza órában	Epoch szám	Training WER (%)	Validation WER (%)
12x1	orosz_rövid	N	8	200	10.5	58.4
15x5	orosz_rövid	I	11.5	100	8.84	41.95
12x1	orosz_közepes	N	18.5	200	7.26	42.56
12x1	orosz_közepes	N	9	100	19.42	50.73
12x1	orosz_közepes	I	8	100	14.53	56.3
15x5	orosz_közepes	N	35	100	10.79	45.6
15x5	orosz_közepes	I	36.5	100	5.63	30.89
12x1	orosz_hosszú	I	27	63	41.93	67.42
15x5	orosz_hosszú	I	45	33	31.91	59.25

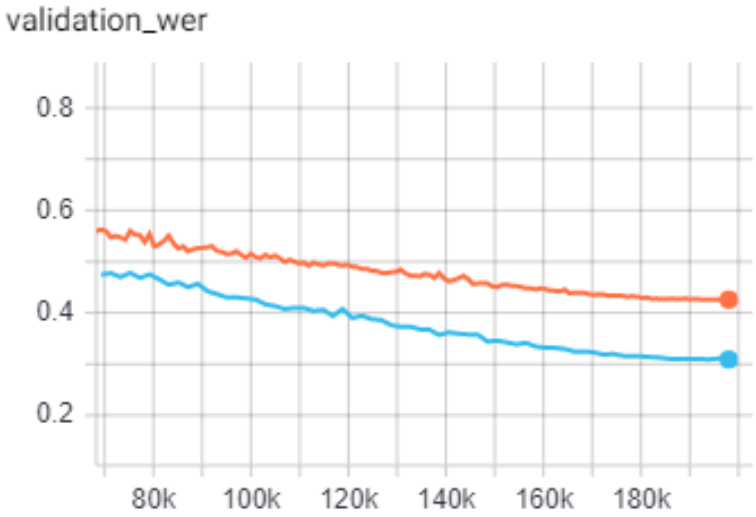
6.1. táblázat. A kipróbált modellek, azok architektúrája és az általuk elért eredmények.

Az összes modellnél a későbbi összehasonlíthatóság végett 0.01-es learning rate-et használtam 0.001-es weight decay-el, CosineAnnealing ütemezővel, melynek 100 bemelegítő lépés volt megadva.

6.1. Különböző architektúrák összehasonlítása

A nagyobb modellek pontosabb eredményeket és alacsonyabb WER-t értek el adatbázistól függetlenül. Viszont ez az idő drasztikus növekedésével járt együtt. 12x1-es architektúra nagyjából 5 millió paramétert, míg a 15x5-ös architektúra 18.9 millió paramétert használt. Futtatási idő esetén ez azt eredményezte, hogy fele annyi epoch-hoz másfélszer annyi időre volt szüksége, ami 3-szoros sebesség csökkenést eredményez. Következtetésképp lehetőség szerint érdemes minél több erőforrással

dolgozni, az idő csökkentése végett és a mélyebb háló útján elindulni a precízebb végeredmény céljából.



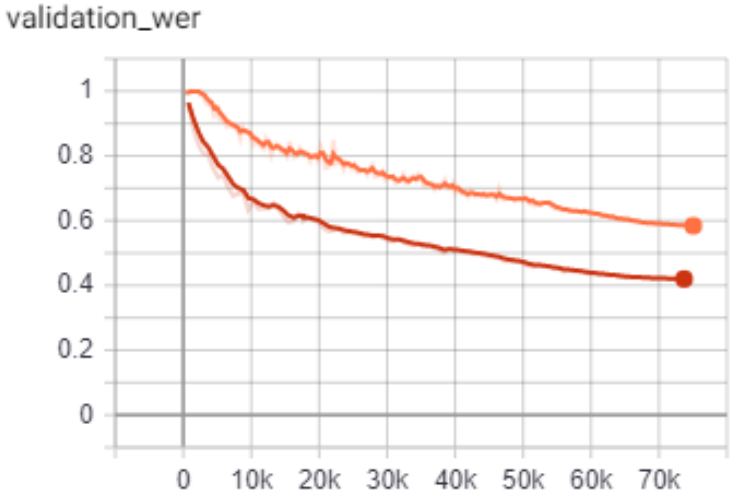
6.1. ábra. 15x5-ös - kék és 12x1-es - narancs architektúrák. Merőleges tengelyen a lépések, batchek száma, függőleges tengelyen a WER értéke.

A 6.1-es ábrán látható viszont a mélyebb, összetettebb háló előnye, hiszen ekvseebb epoch alatt pontosabb eredményeket, alacsonyabb validation WER értéket sikerült elérni vele, mint a 12x1 architektúrával.

6.2. Véletlenszerűen inicializált vs transfer learning

Szinte minden esetben megállapítható, hogy a transfer learning nagyban javította az elért eredményeket, annak ellenére, hogy az átvett modell súlyai egy másik ábécével, másik nyelven lettek tanítva. Látható, hogy az emberek által generált hangok, a nyelvek struktúrája és logikája erősen összefügg.

Kivételt képez az orosz közepes adathalmazon, 12x1-es architektúrával tanított modellek. Erre magyarázat lehet, hogy a kezdeti loss görbéket figyelve a Transfer Learning-et alkalmazó változat kezdetben nem csökkent, 6 epochig. Feltehetőleg ezt egy újabb tanítás indítása megoldhatná, ahol a dekóder megfelelőbb véletlenszerű súlyokat kap inicializálásnál. A training WER-en viszont tettenérhető a javuló csökkenő tendencia.



6.2. ábra. 12x1-es architektúrák validation WER értékei pretrained - vörös és random inicializált - narancs színű súlyokkal. Merőleges tengelyen a lépések, batchek száma, függőleges tengelyen a WER értéke.

6.3. Mozilla Common Voice és Radio2

A két adatbázis fő különbsége az adatok mennyisége volt. A Radio2-ből nagyjából 200 órányi adatot használtam fel, míg a Mozilla Common Voice-ból 45 órányit. Emiatt a Radio2-vel hosszabb tanításokra, de jobb általános eredményekre lehetett számítani.

Ennek megfigyelésére a két különböző adatbázison tanított modellek teljesítményét kell megvizsgálni. A orosz_közepes tanított modell 29%-al jobb WER eredményt ért el az orosz_hosszú adathalmazon tanított modellnél az orosz_közepes-hoz tartozó teszt adathalamzon. Viszont az orosz_hosszú-ból leválasztott teszt adathalmazon lényegesebb, 37% körüli érték volt az eltérés az orosz_hosszú adathalmazon tanult adatok javára. Látható, hogy a nagyobb adatmennyiség egy általánosabban jobban teljesítő modellt eredményezett.

6.4. Javítási lehetőségek

A jobb WER eredmények végett alkalmazhatóak a neurális hálók esetén a gyakran használt augmentációs módszerek. Ezek az elérhető hanganyag mennyiséget dúsítják fel különböző módszerekkel. Ez lehet az eredeti hanganyag torzítása, levágása, zaj hozzáadása vagy sebességének módosítása.

Fontos különbség a megszokott neurális háló alapú beszédfelismerő rendszerekhez képest, hogy nem használtam az implementációmban nyelvmodellt, illetve a legegyszerűbb, greedy, dekódolási eljárással dekódoltam a kimeneti mátrixomat, ami szuboptimális eredményeket nyújt, hiszen ennél összetettebb módszerek szinte kizárólag jobb eredményekhez vezetnek[**?**]. Ettől eltérő dekóder használata javíthat az eredményeken.

Nagyobb adatbázisok alkalmazása jobb általánosítást eredményez, ami a validációs WER érték csökkenésével jár együtt. Érdemes a különböző adatbázisokat összefogni és akár több ezer órányi adaton tanítani.

A sebesség növeléséhez a NeMo toolkit több lehetőséget is kínál. Az egyik a Mixed Precision[1], a másik módszer több GPU bevonása a tanítási folyamatba. Utóbbi esetben több, előre meghatározott GPU közt osztódik el az adat és kerül gyorsabb kiértékelésre.

Lehetséges még a tanítóadat pontos behatárolása. Mivel tanítás közben, a tanítási batch-ekbe különböző hosszúságú hangfájlok kerülnek betöltésre, így például egy 16-os batch méret esetén a GPU hamar kiértékeli az 5 másodperc hosszú hangfájlokat, de nem tud tovább haladni, míg a leghosszabbat, például egy 14 másodperceset nem értékelt ki. Ezen probléma elkerülése, és a GPU jobb kihasználása végett célszerű nagyjából egyforma hosszúságú adatokat felhasználni a tanításhoz.

Természetesen javulást eredményezhet a hiperparaméterekkel való kísérletezés, azok finomítása. Különböző adatok, nyelvek vagy architektúrák más-más paraméterekkel működnek jobban, nincsenek bevett számok, legfeljebb megközelítőleg.

6.5. Végső gondolatok

A beszédfelismerés már régóta velünk van, de egyre inkább elterjed és bekerül a köztudatba, mindennapjainkba a pontosság növekedésének következtében. Ehhez nagyban hozzájárul a klasszikus HMM-től eltérő, napjainkban egyre inkább

[1]Mixed Precision lehetőség magyarázat az NVIDIA oldalán: `https://docs.nvidia.com/deeplearning/nemo/neural_mod_bp_guide/index.html`

felkapott megközelítés a mély neuron háló alapú, end-to-end beszédfelismerés. Egyre több toolkit, és platform jelenik meg, amelyek könnyítik a neurális hálókkal való munkát, azok fejlesztését.

Az angoltól eltérő nyelvű automatikus beszédfelismerő rendszerek fejlesztése és kutatása fontos terület, hiszen a különböző nemzetek polgárai a saját nyelvükön akarnak beszélni. Segítséget nyújthatnak ebben a folyamatban a precíz, angol nyelvű modellek a transfer learning által, aminek következtében drasztikus javulás érhető el az új modell WER pontosságánál.

A nap mint nap megjelenő újabb architektúrák implementálásával a pontosság tovább növelhető, így korábban elképzelhetetlen, az embernél is nagyobb pontosságú beszédfelismerés valósítható meg, mely az élet számos területén használható fel.

Köszönetnyilvánítás

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

Irodalomjegyzék

[1] William Chan – Navdeep Jaitly – Quoc V. Le – Oriol Vinyals: Listen, attend and spell. 2015.

[2] Ronan Collobert – Christian Puhrsch – Gabriel Synnaeve: Wav2letter: an end-to-end convnet-based speech recognition system. 2016.

[3] Evandro Gouvêa: An4 hivatalos oldala (2020. november). `http://www.speech.cs.cmu.edu/databases/an4/`.

[4] Kyu J. Han – Ramon Prieto – Kaixing Wu – Tao Ma: State-of-the-art speech recognition using multi-stream self-attention with dilated 1d convolutions. 2019.

[5] Awni Hannun – Ann Lee – Qiantong Xu – Ronan Collobert: Sequence-to-sequence speech recognition with time-depth separable convolutions. 2019.

[6] Harald Scheidl: Ctc (2018. június 10.). `https://towardsdatascience.com/intuitively-understanding-connectionist-temporal-classification-3797e43a86c`.

[7] Jocelyn Huang, Yang Zhang, Boris Ginsburg, Poonam Chitale.

[8] Rohit Thanki and Surekha Borra: Machine learning in bio-signal analysis and diagnostic imaging (2019). `https://www.sciencedirect.com/topics/engineering/convolutional-neural-network`.

[9] Samuel Kriman, Stanislav Beliaev, Boris Ginsburg, Jocelyn Huang, Oleksii Kuchaiev, Vitaly Lavrukhin, Ryan Leary, Jason Li, Yang Zhang: Quartznet: Deep automatic speech recognition with 1d time-channel separable convolutions. 2019. 10.

[10] Siddhantl: Ctc magyarázata (2019. október 19.). https://sid2697.github.io/Blog_Sid/algorithm/2019/10/19/CTC-Loss.html.

[11] Visual Capitalist: Nyelvek világa (2018. május). https://www.visualcapitalist.com/wp-content/uploads/2018/05/world-of-languages.html.

[12] Wikipedia: Beszéd és hangelismerés idővonala (2020. november). https://en.wikipedia.org/wiki/Timeline_of_speech_and_voice_recognition.

[13] Wikipedia: Mfcc (2020. november). https://en.wikipedia.org/wiki/Mel-frequency_cepstrum.

[14] Wikipedia: Rekurrens neurális háló (2020. november). https://en.wikipedia.org/wiki/Recurrent_neural_network.

[15] Wikipedia: Softmax függvény (2020. november). https://en.wikipedia.org/wiki/Softmax_function.

[16] Wikipedia: Viterbi algoritmus (2020. november). https://en.wikipedia.org/wiki/Viterbi_algorithm.

[17] Neil Zeghidour – Nicolas Usunier – Gabriel Synnaeve – Ronan Collobert – Emmanuel Dupoux: End-to-end speech recognition from the raw waveform. In *Proc. Interspeech 2018* (konferenciaanyag). 2018, 781–785. p. URL http://dx.doi.org/10.21437/Interspeech.2018-2414.