SZAKDOLGOZAT

Klöczl Ádám

2022

Budapesti Corvinus Egyetem

Gazdálkodástudományi kar

Számítástudományi Tanszék

Hangfelismerés magyar nyelven

Készítette: Klöczl Ádám

Gazdaságinformatikus szak

Dr. Mohácsi László

2022

Tartalomjegyzék

1. Bevezetés

A beszédfelismerő rendszerek fejlődése az elmúlt években látványos fejlődésen ment keresztül. A legtöbb szolgáltató a megoldását online felhőalapú szolgáltatásként kínálja, amelyet használat alapú díjért cserébe lehet igénybe venni. Rövidtávú megoldásnak megfelelő lehet, de abban az esetben, ha egy szolgáltatásnak az alappillére egy beszédfelismerő rendszer és annak funkciói, akkor az externális módszer hosszú távon költségesebb és kockázatosabb lehet, mint egy saját megoldás megvalósítása. Nem is beszélve arról a tényről, hogy ennek az igénybevételéhez folyamatos stabil internetkapcsolatra van szükség. Sajnos limitáltak a lehetőségek egy saját megoldás megvalósítására és ebben az esetben a betanító adatokat meg kell vásárolnia a fogyasztónak, vagy elő kell állítania.

Emellett azt is fontos megemlíteni, hogy a legtöbb jól működő beszédfelismerő rendszer csak angol és egyéb jobban elterjedt nyelveken érhetők el. A magyart a beszélők alacsony száma és a nyelv komplexitása miatt nem egyszerű és jutalmazó az ilyen rendszerekbe beintegrálni, ezért nem is olyan széles a választék, ha ilyen szoftvert keresünk.

Szerencsére a Mozilla kifejlesztett egy olyan nyílt forráskódú Speech-To-Text motort, amely elviekben képes bármely beszélt nyelvet szöveggé alakítani abban az esetben, ha rendelkezünk megfelelő mennyiségű és minőségű adattal a betanításhoz. A DeepSpeech megoldást tudna nyújtani az előbb említett két problémánkra, ha valóban képes egy pontos beszédfelismerő modell generálására.

A dolgozat célja megvizsgálni ennek a nyílt forráskódú Speech-To-Text motornak a teljesítményét a Mozilla Common Voice adathangtár betanításával.

Abban az esetben, ha egy átlagos felhasználó szeretne hozzáférni Speech-To-Text szoftverhez magyar nyelven, akkor az alábbi alternatívákra vannak lehetőségei:

- Google: A Google az évek alatt tökéletesítette a beszédfelismerő szoftverét hiszen a saját fordítójukban és keresőmotorjukban is ez az egyik fő funkció. Emellett a térkép alkalmazásukba is bele van integrálva, ahol rendkívül pontosan működik. A Cloud Speech-To-Text szolgáltatásukat be lehet integrálni a helyi rendszerekbe.
- Online Diktálás: Az https://onlinediktalas.hu/ oldalba épített beszédfelismerő szoftver is hihetetlenül pontosan konvertálja át a megadott magyar hanganyagot szöveggé. A kezelőfelülete letisztult és könnyen kezelhető, ám nem reszponzív a weboldal, vagyis a mobillal való kezelés nagyobb nehézséget fog okozni.

A dolgozat a következő négy fő szempontot tárgyalja:

- Elméleti háttér: Egy rövid bevezetés a mesterséges intelligencia történetéről, főbb fogalmairól és használatáról. Szó lesz emellett még a hangfelismerésről és a DeepSpeech-ről is.
- Adattisztítás: Egy Javascript algoritmus megírása, mely eltávolítja a speciális karaktereket, egységesíti a szövegeket és a megfelelő formátumban menti le az adatokat.
- Betanítás: A TensorFlow-ra épült DeepSpeech beszédfelismerő modell betanítása a Mozilla Common Voice hangmintatár alapján.
- Tesztelés: Az elkészült modell próbája és értékelése különböző szempontok alapján.

A modell betanítása az alábbi 4 lépésből áll:

1. Fejlesztői környezet létrehozása
    1) Dependenciák telepítése
    2) Hanganyagok formátumának konverziója
2. Adattisztítás
    1) Speciális karakterek eltávolítása és kisbetűssé alakítás
    2) Kiírás csv típusú fájlba
3. A DeepSpeech algoritmus betanítása
    1) Dependenciák telepítése
    2) Betanítás
4. A modell tesztelése
    1) Teszt környezet létrehozása
    2) Tesztek végrehajtása
    3) Értékelés

Rendkívül fontos az alapos tesztelés, hiszen a dolgozat alapvető célja a DeepSpeech algoritmus pontosságának ellenőrzése. Ezért több iteráción keresztül történik a tesztelés, amely mind férfi és női hangot alkalmazó próbákat tartalmaz.

A szakdolgozat megírása olyan formátumban történt, hogy bárki képes legyen betanítani saját magának a DeepSpeech-et bármilyen Mozzilla Common Voice nyelvcsomaggal. **A letöltött források, forráskódok, lépések, modell, excel, összesített erdemények és minden releváns fájl elérhetőek ezen a linken**.: https://github.com/AdamK18/szakdolgozat

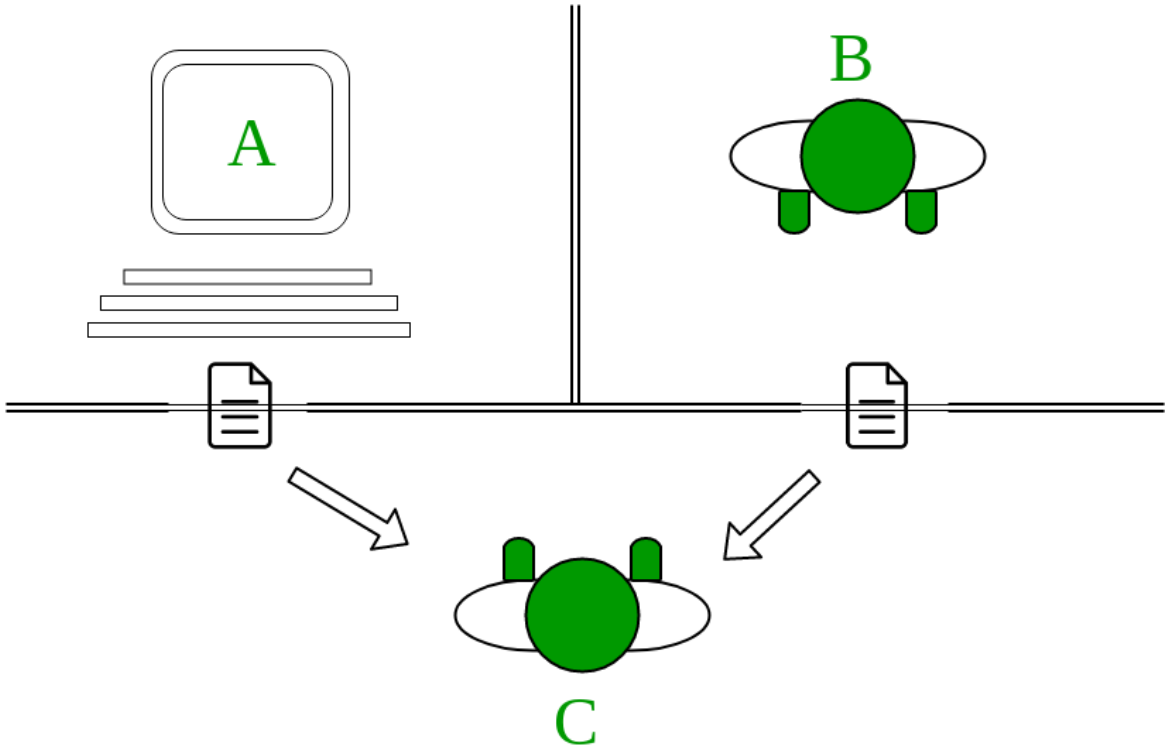
2. A mesterséges intelligencia

2.1 A mesterséges intelligencia rövid története

Másnéven AI, az angol artificial intelligence szó rövidítése, korunk egyik legmeghatározóbb és legérdekesebb technológiája. Az AI képes komplex feladatokat pontosan végrehajtani, döntéseket meghozni, előrejelzéseket készíteni adatokból és fejleszteni magukat. Már az 1950-es évek óta foglalkoztatja a kutatókat, hogy milyen módon lehetne megtanítani a számítógépet racionálisan vagy emberhez hasonlóan gondolkodni (WAI, 2021).

Alan Turing tette fel a kérdést, hogy: „Képesek-e a számítógépek gondolkodni?” (Turing, 1950), ami elindította az AI-t a fejlődés útján. Turing egy angol matematikus volt, aki a második világháborúban segítette a szövetséges hatalmak győzelmét az ENIGMA kód feltörésével. A háború vége után kevesebb mint 10 évvel Turing megírta tudományos cikkét: COMPUTING MACHINERY AND INTELLIGENCE (WAI, 2021).

Turing megfogalmazta, hogy milyen akadályokat kell átugrania a mesterséges intelligenciának, hogy valóban intelligensnek hívhassuk. A tesztek egyike az úgy nevezett imitation game. Ebben a módszerben egy ember egy számítógéppel és egy másik személlyel írásban üzeneteket váltanak egymás között. A cél az, hogy az első személy kiderítse, melyik fél a számítógép (Turing, 1950).



1. ábra Turing Test – forrás: Bansal (2018) - https://www.geeksforgeeks.org/turing-test-artificial-intelligence/

Az üzenetírás lehetővé teszi, hogy az ellenőrző válaszát ne befolyásolja bármilyen emberi jellem, mint a kinézet, amiben a számítógép nem versenyezhet, csak az íráskészség legyen a döntő faktor. A teszt teljesítéséhez minimum 30%-os győzelmi aránnyal kell rendelkeznie a számítógépnek (BBC, 2014).

Ezt a tesztet először 2014-ben hajtották végre sikeresen, amikor az AI 33 százalékos sikerrel hitette el a zsűrivel, hogy ő egy 13 éves ukrán fiú (BBC, 2014).

2.2 A mesterséges intelligencia típusai

Az alábbi mesterséges intelligencia típusokat különböztetjük meg:

- Narrow AI: Másnéven Gyenge AI. Ezt a fajta MI-t azért hívjuk szűknek, mert a kontextus, amiben dolgoznak relatívan limitált a testvéreihez képest. Egy feladatra lettek kiképezve, hogy azt rendkívül jól végezzék. Ilyenek például a kép és beszédfelismerő algoritmusok, keresőmotorok vagy az úgy nevezett személyes segédek, mint az Apple által tanított Siri. Ez a legmagasabb szint, amelyet az emberiség eddig sikeresen elért. Működésekből hiányzik a tudatosság és az érzelem (Microsoft, 2021a).
- Artificial General Intelligence: Erős AI-ként is nevezik, mert elméletben képes megoldani bármilyen hozzácsatolt feladatot, tudatos gondolatokkal, kreativitással és képzelettel rendelkezne, mint egy ember. Ám egy ilyen sokoldalú entitás elkészítése és betanítása még meghaladja a mai emberi kapacitásokat (Microsoft, 2021a).
- Artificial Super Intellgience (ASI): „Egy szuperintelligenciával rendelkező számítógép képes lenne az embert csaknem minden területen túlszárnyalni, többek között a tudományos kreativitásban, az általános bölcsességben és a társadalmi készségekben is” (Microsoft, 2021a).

2.3 A Narrow AI típusai

Ezek mellett megkülönböztetjük a szűk AI-nak két típusát, a gépi tanulást és a mély tanulást:

- Machine Learning: Betanítása adathalmazokkal történik. Az algoritmus azonosítja a mintákat és adatmodelleket készít, amikből előrejelzéseket képes végrehajtani. Ha a modell pontosan és céljainknak megfelelően lett betanítva, akkor elérte a Narrow AI szintjét (WAI, 2021).
- Deep Learning: A gépi tanulás egy fejlettebb fajtája. Hasonlóan itt is nagy adathalmazból tanítják be, ám úgy nevezett neurális hálót épít ki az algoritmus

hasonlóan az emberi agyhoz. A háló több réteggel rendelkezhet és minél több a réteg annál komplexebb a modell. Ez azért szükséges, hogy ki legyen alakítva a kapcsolat az adatok között. Ezzel a tanulási fajtával képes megtanítani az ember a számítógépet a képek felismerésére (WAI, 2021).

3. A gépi tanulás

3.1 A gépi tanulásról bővebben

A Machine Learning a mesterséges intelligencia egy részhalmaza, amely adatokból képez matematikai adatmodelleket. A gépi tanulás algoritmus ezekből a modellekből képes előrejelzéseket végezni, amely könnyen alkalmazkodik a változó helyzetekhez.

Alkalmazásának területei:

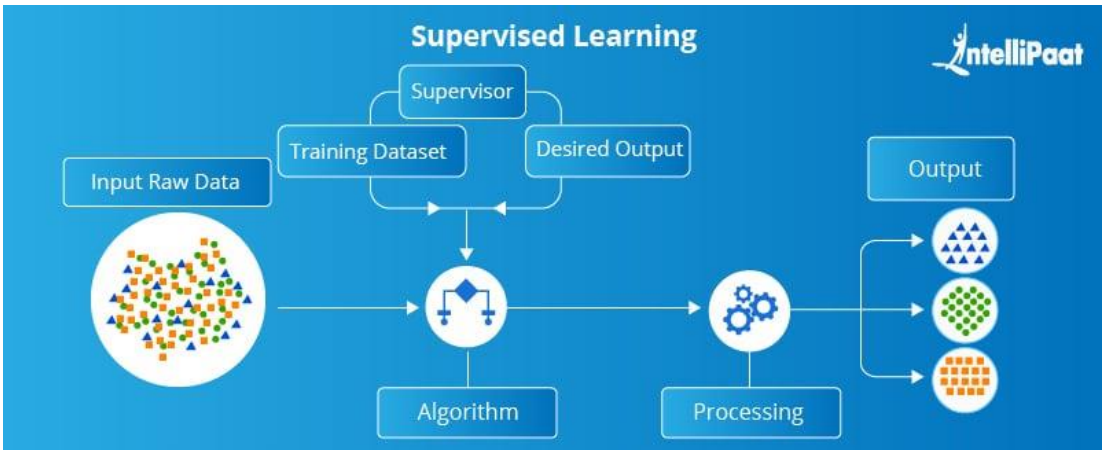
- Elemzés: Használatával könnyen azonosíthatjuk a strukturált vagy strukturálatlan adathalmazokban a mintákat vagy szerkezeteket, amivel pontosíthatunk az előrejelzéseinken (Microsoft, 2021b).
- Felhasználói élmény fejlesztése: A ML segíthet a felhasználók általános elégedettségük növelésében személyre szabott reklámokkal, célzott tartalommal, chatbotokkal vagy akár virtuális asszisztensekkel is. Ez bizonyítottan növeli a visszatérő ügyfelek számát és a vállalat teljesítményét (Microsoft, 2021b).
- Kockázatcsökkentés: A virtuális átverések, vagy angolul scam-ek, hatalmas veszélyt jelentenek az óvatlan felhasználókra. Az ellopott adatokkal visszaélhetnek a kártevők, zsarolhatnak, rongálhatnak vagy kereskedhetnek velük. Ezért rendkívül fontos a digitális védelem kialakítása. A gépi tanulás képes felismerni a különböző scam-eket, ugyanis felépítésük mindig hasonló marad (Microsoft, 2021b).
- Ügyfelek viselkedésének előrejelzése: Az említettek alapján a gépi tanulás tökéletes a meglévő adatokból előrejelzések készítésére. Ezt akár a felhasználók viselkedésének elemzésére is fel lehet használni. Megjósolható az aktivitásuk, illetve a meglévő adatok bányászásával optimalizált termékjavaslatokat lehet készíteni (Microsoft, 2021b).
- Szűrés: Hasonlóan a kockázat csökkentéshez, az algoritmus használható a kevésbé érdekes tartalmak szűrésére és ignorálására. Mivel e-maileket bárki küldhet bármikor, ezért hamar megtelítődhet a digitális postaládánk hulladékokkal. Ezt a folyamatot spam szűrésnek is hívjuk (Microsoft, 2021a).

- Anomáliák észlelése: Ha a ML használható arra, hogy az adatok között kapcsolatokat és szerkezeteket észleljen, akkor az ellenkező esetben is működik, ha például valami nem illik a halmazba. Ehhez a legjobb példa a Facebook és Gmail által használt bejelentkezés ellenőrzés. Ha a rendszer azt érzékeli, hogy a felhasználó a szokásostól eltérő fizikális helyről és eltérő időben jelentkezik be, akkor további megerősítésre kéri a bejutni kívánó személyt. Ez rendkívül hasznos tud lenni profilunk megőrzésében, ha esetleg rosszindulatú felhasználók próbálnának hozzáférni (Microsoft, 2021a).
- Automatizáció: Mivel a machine learning rendkívül precízen végzi el a jól definiált feladatokat, ezért a segítségükkel automatizálni is lehet a különböző folyamatokat. Ez lehetővé tenné a vállalatoknak, hogy bizonyos feladatokat azonnal és pontosan elvégezzenek, ami az emberi beavatkozás szükségét is felszámolja.
- Képfelismerés: A különböző algoritmusok képesek különböző tárgyakat vagy embereket felismerni, attól függően mire lettek betanítva.
- Hangfelismerés: Egy alaposan betanított ML algoritmus felismeri a beszélt szöveget, képes akár értelmezni is azt.

3.2 A Machine Learning algoritmusok betanítása

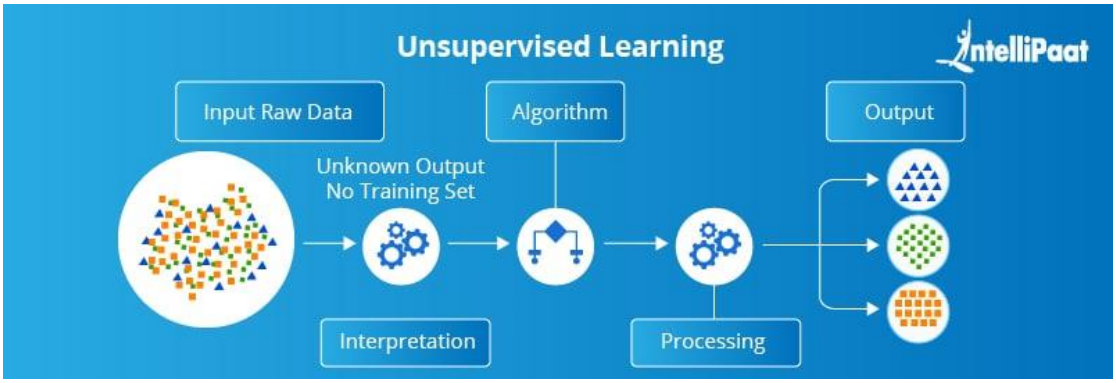
A gépi tanulás algoritmusoknak a betanítása 3 különböző módszer használatával érhető el. Ezeknek a lényege, hogy az adatokat ne vakon értelmezze a ML algoritmus, minimális felhasználói útmutatás szükséges a maximális pontosság eléréséhez.

- Felügyelt tanulás: Felügyelt tanulás, vagy angolul Supervised Learning alatt azt értjük, hogy a tanításhoz használt adathalmaz mellett az algoritmusnak a kimenetet is tápláljuk, így saját maga ki tudja alakítani a kapcsolatokat a feladat és a megoldás között. Ha egy képfelismerő algoritmusnak különböző képeket adnánk macskákról és kutyákról a megfelelő címkékkel, akkor az algoritmus idővel felismerné a tulajdonságait egy kutyának vagy fordítva. Így ha később egy nem címkézett képet adnánk a felügyelt tanulás algoritmusnak, akkor különböző százalékú bizonyosságokkal el tudná dönteni, hogy a kép melyik állatot ábrázolja. Ez azért nagyon fontos, mert ezt a logikát bármilyen más területen is használni tudjuk, mint például hangfelismerésben, ami dolgozat szempontjából releváns (SUR, 2019).



2. ábra Supervised Learning - Forrás: SUR (2019) - https://intellipaat.com/blog/supervised-learning-vs-unsupervised-learning-vs-reinforcement-learning/

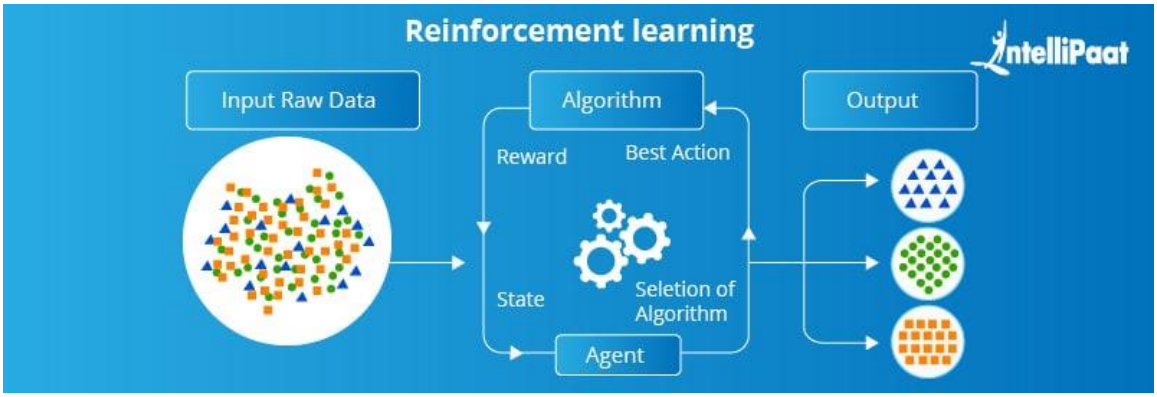
- Felügyelet nélküli tanulás: A felügyelet nélküli tanulás, vagy angolul Unsupervised Learning az előbb említett testvérének egy bizonyos ellentétje. Ennél a módszernél az adathalmazhoz nem csatolunk output-ot, vagyis nem címkézzük meg, hogy melyik adat milyen értékkel rendelkezik. Ilyenkor az algoritmus megpróbálja analizálni, majd hasonlóságok alapján csoportosítani az inputokat. Például amelyik képen csőr található, azokat egy csoportba helyezi, amiknek füle van, azokat másikba. Ezek alapján nem tudná megítélni, hogy melyik adatcsoportba melyik állatot kategorizálta, csak annyit, hogy az egyedei között van egy bizonyos hasonlóság. Viszont, ha a programozó tippekkel látta el az algoritmust, ha például ez a csoport csőrökkel rendelkezik, akkor az egy madár, akkor máris képes a csoportok felcímkézésére (SUR, 2019).



3. ábra Unsupervised Learning - Forrás: SUR (2019) - https://intellipaat.com/blog/supervised-learning-vs-unsupervised-learning-vs-reinforcement-learning/

- Megerősítő tanulás: A megerősítő tanulás, vagy angolul Reinforcement Learning különbözik a legjobban a 3 tanulói módszer közül. Az algoritmusnak nem adnak semmilyen útmutatást, hogy hogyan és milyen eredményt kellene elérnie, csak azt

érzékeli, hogy ha x tevékenységet végez, akkor jutalmat kap, ha y-t, akkor negatív jutalmat. Hasonlóan lehet erre gondolni, mint egy videójátékra. A kezdő játékos nem tudja, hogyan és mit kell csinálni, csak azt, ha bizonyos módszerekkel egyre messzebbre jut az akadályokkal teli pályán, akkor jutalmat kap pontok szerepében, ha figyelmetlenül belefut egy akadályba, akkor újrakezdődik a játék. Dióhéjban ugyanígy működik a megerősítő tanulás, ahol jutalmazva van az algoritmus a sikeres teljesítésért. Hasonlóan működik az Elon Musk által alapított OpenAI, ami egy reinforcement learning tanító keretrendszer (SUR, 2019).



4. ábra Reinforcement Learning – Forrás: SUR (2019) - https://intellipaat.com/blog/supervised-learning-vs-unsupervised-learning-vs-reinforcement-learning/

4. Hangfelismerés

4.1 A hangfelismerés fejlődése és történelme

A beszédfelismerés történelme egészen az 1950-es évekig nyúlik vissza. Ekkor még nagyon minimális funkciókkal készültek el az első szoftverek, hiszen ekkor még maga a számítógépes technológia is rendkívül kezdetleges volt. Az első készüléket a Bell Laboratories készítette 1952-ben, amely csak számokat volt képes megérteni egyesével diktálva egyetlen egy személytől. Tíz évvel később, 1962-ben, az IBM által létrehozott Shoebox nevű algoritmus már 16 angol szóból álló szókinccsel rendelkezett. Érdemes hangsúlyozni a tényt, hogy a két találmány között 10 év eltérés volt. Egyik szoftver se általános használatra, hanem specifikus szavakra lett beprogramozva és csak egy forrásból értette meg az utasításokat, ami nagyban megkülönbözteti a mai rendszerektől. (SHOSR, 2022)

Az 1970-es években Carnegie Mellon szoftvere, amelyet Harpy-nak neveztek el és a Speech Understanding Research program keretei között készült, képes volt több, mint 1000 szó megértésére. Emellett a Bell Laboratories tovább fejlesztette a beszédfelismerő rendszerét és

már képes volt több személy hangját is értelmezni. Az 1980-as évek áttörése a Hidden Markov Model megjelenése volt. Ez egy olyan statisztikai módszer, amely az egyszerű szavak és hangminták keresése helyett százalékos valószínűséggel próbálta kitalálni, hogy a hang milyen szónak feleltethető meg. (SHOSR, 2022)

A személyi számítógépek terjedése és a processzorok fejlődése mind szerepet játszottak a hangfelismerés fejlődésében is. A 2000-es évek elejére a beszédfelismerés pontossága már elérte a 80 százalékot. Az igazán nagy ugrás a Google Voice Search megjelenése volt, amely az átlagemberhez is eljuttatta a lehetőséget, hogy kipróbáljon egy felismerő szoftvert. A begyűjtött adatok segítségével tovább tudták fejleszteni a saját rendszerüket. (SHOSR, 2022)

Mára a digitális segédek egyszerűen megértik a parancsokat, amiket az angolul beszélők mondanak nekik. A következő lépés a technológia kiterjesztése a világ többi nyelvére.

4.2 A hangfelismerés működése

Annak érdekében, hogy a telefonunk vagy egyéb eszközünk megértse, amit mondunk neki, számos matematikai és algoritmikai folyamaton kell átesnie a szövegnek. Vegyük például a különböző személyes segédeket, vagy angolul voice assisstant-eket, amik képesek a hangalapú utasításokat átkonvertálni a digitális megfelelőjükre. Például: Siri, Alexa, Cortana, Google Voice Assisstant és egyéb más. Mind hasonló lépésekben működnek és hasonló eszközöket vetnek be, hogy megértsék az emberek szándékait. Ezeket a lépéseket röviden fogom jellemezni az alábbi pontokban (HDSW, 2021).

1. Először is az algoritmus feltördeli a nyers hangot kisebb részekre. Ezt a szóvégi hangerő és magasság mérésével teszik. Így a szoftver sikeresen meg tudja különböztetni a szavakat egymástól és egyesével képes értelmezni azokat.
2. Majd a digitális tömbön különböző algoritmusokat hajtanak végre, amikkel elemzik és összehasonlítják az inputot a saját adatbázisukkal.
3. Ezután százalék alapúan a legjobban egyező szóval párosítják és végül összefűzik azokat.

Ezek után feltehetünk magunknak pár kérdést a működésével kapcsolatban. Hogyan képes kiszűrni a háttérzajt a beszédből? Hogyan képes megérteni a világ különböző részén élő emberek dialektusát? Biztosan azokat a szavakat adta vissza, amelyeket kimondtunk? Értelmes egyáltalán a mondat, amit visszaadott? Ezek valós kérdések, amik nagyon fontosak az algoritmus helyes működése érdekében és amelyeket a mérnökök is feltettek egymásnak a

tervezésekor. Az alábbi akadályokat kellett figyelembe venniük a hangfelismerés fejlesztésekor (HDSW, 2021).

- A hangmagasságok és erősségek változóak a felhasználó hangulatától függően. Ha ideges az ember akkor gyorsabban és hangosabban kommunikál, ugyanakkor, ha szomorú akkor lassabban és halkabban. Az algoritmusnak képesnek kell lennie ezekre az ingadozásokra való megfelelő reagálására, hogy rendesen megértse a felhasználó szándékát (HDSW, 2021).
- A háttérzajok kiszűrése talán még nagyobb akadályt jelent, mint a váltakozó hangulat. A kocsik zúgása, hirtelen csattanások vagy folyamatos alacsony hatású torzítás mind szerepet játszanak a kimenetelben. Az algoritmusnak képesnek kell lennie elkülöníteni a felhasználó hangját a háttérhatásoktól, különben garantált lesz a félreértés.
- A háttérzajok mellett kihívást jelenthet még a felhasználó beszéde is. Mivel a világ hatalmas és számos különböző kultúra és dialektus létezik, ezért nem mindenki fogja ugyanolyan hangnemben feltenni például azt a kérdést, hogy: „Can I have some water please?”. Ez eltér az előbb említett problémáktól abból a szempontból, hogy a kiszűrt kulcsszó teljesen más kiejtéssel rendelkezhet attól függően, hogy milyen etnikai háttérből származik a felhasználó. Egy amerikai angol és egy brit angol különbözően ejtik ki a „water” szót, amit az algoritmus is teljesen máshogyan kezel. Ezért gyakran különböző akcentusokra készítenek egy iker algoritmust, ami kifejezetten arra a kiejtésre van specializálva. Szerencsére nekünk magyaroknak a kiejtéseink között szignifikánsan sok különbség nem észlelhető, mivel a nyelvet beszélők száma 10-20 millió között állapítható meg. Ezért a lehetséges akcentusok száma relatívan kicsi a többi nyelvhez képest. Ehelyett inkább a területi eltérések különböző szókincs használatot eredményezhetnek, amit szintén tárolnia kell egy adatbázisnak. Sajnos a nyelvet beszélők alacsony száma miatt ma a legtöbb hangutasítással rendelkező eszköz nem támogatja a magyar nyelvet, de már léteznek beszédfelismerő algoritmusok, amelyek pontosan képesek megérteni a magyar nyelvet (HDSW, 2021).
- Az utolsó lépés a szöveg átkonvertálásában az, hogy valóban van-e értelme, nincsenek-e nyelvi hibák, értelmetlen kifejezések benne. A fejlettebb szoftverek képesek értelmezni a mondatokat egy bizonyos kontextusban, és ahhoz képest átalakítani a szavakat, ha azok egyeznek egy másikkal és jobban beleillenek(-e) értelem szempontjából. Például, ha a felhasználó azt mondja, hogy „nagy baj van” és közben háttérzaj torzítja a hangot, akkor a visszaadott szöveg lehet az, hogy „nagy vaj van”. De

a fejlett beszédfelismerő érzékeli, hogy van egy hasonló szó a vaj helyett, ami jobban illeszkedik az értelem környezetébe, így kicseréli baj-ra (HDSW, 2021).

- Ha voice assisstant-nak adunk utasítást, akkor a már átkonvertált és értelmezhető szöveget átalakítja utasításokká. Ezt neurális hálók segítségével teszi meg, amiben különböző kulcsszavak különböző jelentésekhez és utasításokhoz vannak kapcsolva.

4.3 A hangfelismerő szoftverek célja

Hasonlóan a legtöbb technikai fejlődéshez, ennek is a fő célja az előrehaladásban, hogy megkönnyítse az emberek mindennapi életét. Lehetővé tenné, hogy multitask-oljunk, ha esetleg a helyzet nem engedné, írhatnánk és használhatnánk dolgokat, ha azokra eddig fizikális korlátozás miatt nem volt lehetőségünk és megkönnyítené bizonyos feladatok elvégzését, ha rendelkezünk a megfelelő eszközökkel:

- Voice Assistants: Az olyan segédeszközök, mint Alexa és Siri automatizáltak bizonyos tevékenységeket, amiket az átlag felhasználó eddig lassabban vitt végbe. Itt olyan egyszerű tevékenységekről van szó, mint gyors információgyűjtés az internetről, időjárás előrejelzés megtekintése, saját tennivalólista készítése vagy akár egy mondat más nyelvre történő lefordítása.
- Multitasking: A beszédfelismerés lehetővé tenné a bizonyos eszközök használatát miközben mindkét kezünkre szükségünk van, vagy esetleg biztonsági okokból nem használhatjuk. Például vezetés közben zene váltás, Gps beállítás vagy főzés közben egy recept ellenőrzése. Ezek mind olyan dolgok, amik eddig a teljes figyelmünket igényelték volna és potenciális balesethez vezetett volna a hanyagolásuk.
- Fogyatékkal élők segítése: Legtöbbünk teljesen biztosra vette, hogy bármikor hozzáférhet a telefonjához vagy számítógépéhez és utánanézhetnek dolgoknak. Ám azoknak, akik fizikailag korlátozva vannak nagy kihívást jelenthet az ilyen tevékenységek véghezvitele. Ilyen esetekben a hangvezérlés egy tökéletes funkció volna a fogyatékkal élők és mozgássérültek segítésére.

4.4 A DeepSpeech röviden

„A DeepSpeech egy nyílt forráskódú Speech-To-Text motor, amely egy gépi tanulási technikákkal kiképzett modellt használ, Baidu Deep Speech kutatási cikke alapján. A DeepSpeech projekt a Google TensorFlow szoftverét használja a megvalósítás megkönnyítésére” (DPSCH, 2021).

A DeepSpeech lényegében egy olyan modell, amit a megfelelő hangmintatár segítségével betaníthatunk, hogy megalkossuk a saját beszédfelismerő szoftverünket. Sajnos a betanításra csak Pythonnal van lehetőségünk Linux-on, de egy kész modell használata több operációs rendszeren is elfut és a lehetséges programozási nyelvek száma is megnő:

- Python
- C
- C#
- Java
- Javascript

A hangmintatárat a Mozilla Common Voice projektje adja, amiben több, mint 20 órányi magyar hanganyag található és kifejezetten beszédfelismeréssel rendelkező alkalmazások betanítására tervezték.

4.5 Mozilla Common Voice

„A Mozilla Common Voice egy kezdeményezés arra, hogy segítsük a gépeket megtanítani, hogyan beszélnek az emberek” (CMN, 2022). Ez a nyílt hozzáférésű adatbázis az egyik alappillére a dolgozatnak. A körülbelül 20 órányi validált magyar hanganyag különböző méretű részekre lesz felosztva és betáplálva a DeepSpeech modellbe, hogy letesztelhessük milyen pontosságú magyar beszédfelismerésre lesz képes. A kezdeményezés megálmodói szeretnék, hogy bárki szabad kézzel építhessen hasonló megoldásokat, mert szerintük a nagyvállalatok hangmintatárának hozzáférhetetlensége akadályozza az innovációt. Bárki részt vehet a hangjának adományozásával, vagy a többi donor hozzájárulásának ellenőrzésével (CMN, 2022).

A DeepSpeech pontos betanításához több ezer óra hanganyag szükséges. A Common Voice angol mintatára 2200 órányi anyaggal rendelkezik. Ez bőven elég lenne egy hatékony modell kiképzésére. Ha szeretnénk, hogy a jövőben több magyar hangfelismeréssel foglalkozó kezdeményezés szülessen, akkor támogathatjuk a Common Voice munkáját adatok hozzájárulásával vagy ellenőrzésével. Ezt a webes felületen tehetjük meg és napi pár percnyi foglalkozással nagyban hozzájárulhatunk az adatbázis gyarapodásához.

A továbbiakban a modell betanításához szükséges lépéseket fogom részletezni.

5. Dependenciák telepítése

A DeepSpeech futtatása előtt pár követelménynek meg kell felelnünk. Pontosabban, le kell töltenünk különböző szoftvereket, amelyek közül némelyik operációs rendszertől függetlenül szükséges. Az olyan függőségeket, amik a modell telepítéséhez, betanításához és használatához feltétlen szükségesek, dependenciának hívjuk a továbbiakban így fogok hivatkozni rájuk a dolgozatban. Ezek különböző eszközök, amelyek segíteni fognak bennünket a célunk elérése érdekében, vagy nagyban megkönnyítik a munkánkat.

5.1 Git

A Git egy olyan szoftver, amely lehetővé teszi, hogy kezeljük az alkalmazásunk vagy dokumentumaink különböző verzióit, vagyis visszaállhatunk egy régebbi állapotra, ha hibába ütköznénk. Ez rendkívül hasznos tud lenni, főleg akkor, ha az alkalmazás vagy rendszer, amin dolgozunk nem követi a lépéseinket bezárás után, mint például a Word. Emellett a csapatmunkát is nagyban leegyszerűsíti a munkaszálak összefűzésével. De a fő célja a betanítás érdekében, az, hogy leklónozzuk a DeepSpeech repository-t GitHub-ról. Repository-nak hívjuk azokat az állományokat, amelyeket a Git követ és a clone paranccsal könnyedén letölthetjük ezeket a rendszerünkre. A GitHub egy webtárhely, amely különböző repository-kat tárol, hogy fejlesztők által könnyedén hozzáférhető legyen.

Töltsük le a git-et az alábbi linkről (https://git-scm.com/downloads) és telepítsük azt. Egy command line megnyitásával és az alábbi parancs segítségével ellenőrizhetjük a letöltött Git verzióját és megbizonyosodhatunk arról, hogy megfelelően fel lett telepítve.

```
git --version
```

A Git működése első találkozásra furcsának vélhető, de rövid ismerkedés után könnyen megtanulható. Úgy nevezett commit-ok segítségével lépésekben rögzíti a megfigyelt állomány változásait, hogy az könnyen visszafordítható legyen. Egy commit lényegében egy snapshot ami a legutóbbi változtatásokat tartalmazza. Annak érdekében, hogy legyen egy átfogó elképzelésük a Git használatával kapcsolatban, röviden bemutatok pár Git parancsot, amelyek a használatának az alapját képzik és szükségesek, ha precizitást igénylő munkát végzünk.

```
git init
```

Létrehoz egy repository-t.

```
git clone
```

Ennek a parancsnak a segítségével könnyedén lemásolhatunk egy repository-t lokális használatra. A fájlok mellett letölti a repository teljes változástörténetét is.

```
git add (--all)
```

Ennek a parancsnak két használata is van. Alapvetően nem minden fájl állapota van megfigyelve és ahhoz, hogy követhessünk egy vagy több tetszőleges fájlt, az add parancsot kell használnunk. Emellett ennek a command-nak a használatával hozzáadhatjuk az uncommitted fájlokat a staged csoportba. A zárójellel körbeírt „—all” egy úgy nevezett paraméter, amivel az összes fájlra kiterjed a parancs hatóköre. Legtöbbször ezzel a paraméterrel együtt használjuk.

```
git commit (-m „üzenet”)
```

A staged fájlok kommittolására használjuk. Ezzel egy új snapshot-ot hozunk létre amire bármikor tetszőlegesen visszatérhetünk. Ha nem használtuk az add parancsot a megváltoztatott fájlokra, akkor nem tudjuk használni a commit parancsot, hiszen az csak a staged fájlokra érvényes.

```
git push
```

A commit-ok feltöltése a forrásba, ami lehet GitHub-on, BitBucket-en vagy bármely repository tároló webhelyen.

5.2 Nvidia

Videókártyánk használatának érdekében szükséges letöltenünk és telepítenünk a legfrissebb Nvidia driver-eket, hogy ne ütközzünk nem várt problémákba a modell használata közben. Töltsük le az Nvidia honlapjáról (https://www.nvidia.com/en-us/geforce/drivers/), majd telepítsük azt. Ez egy hosszabb folyamat lesz és közben lehet újra kell indítanunk számítógépünk.

5.3 Windows Subsystem for Linux 2

Linux vagy MacOs-vel rendelkezők nyugodtan kihagyhatják ezt a lépést. Míg a modellt le lehet futtatni nagyjából az összes elterjedt operációs rendszeren, betanítani a DeepSpeech-et csak is a fent említett két operációs rendszeren lehetséges. Ezért a Windows-os felhasználók kénytelenek lesznek egy külön megoldást választani és futtatni egy szabadon választott Linux disztribúciót a rendszerükön. Szerencsére a Microsoft nagyban megkönnyítette a munkánkat a WSL folyamatos fejlesztésével, ami lehetővé teszi, hogy egy két gyors telepítés után már használhassuk is a beépített Linux disztribúciónkat bármilyen óriási változtatás vagy hozzáértés nélkül.

Nyissunk egy Command Prompt-ot majd gépeljük be az alábbi parancsot:

```
wsl –install
```

Ezután egy újraindítás mindenképpen szükséges, hogy a Windows véglegesítse a telepítést. Újraindítás után használjuk az update parancsot, hogy frissítsük a kernel verzióját.

```
wsl –update
```

Ezután az alábbi parancs segítségével alapértelmezetté tehetjük a frissebb kettes verziót.

```
wsl –set-default-version 2
```

5.4 Ubuntu

A WSL telepítése önmagában nem elég, szükségünk lesz egy disztribúcióra is. Ezt úgy képzeljük el, mintha egy számítógépen nem lenne feltelepítve semmilyen operációs rendszer és úgy szeretnénk használni. A legegyszerűbb, ha az Ubuntu 18.04-es verzióját telepítjük. Ez a verzió a legjobban kompatibilis az összes olyan dependenciákkal amiket még a későbbiekben telepíteni fogunk. A letöltés és telepítés egy rendkívül egyszerű folyamat, szimplán nyissuk meg a Microsoft Store-t és keressünk rá az Ubuntu 18.04-re.

Telepítés után gépeljük be az alábbi parancsot, hogy a WSL második verzióját használja alapértelmezetten.

```
wsl –set-version Ubuntu 2
```

Mivel ez egy külön rendszernek minősül, ezért a legtöbb driver-t külön le kellene tölteni ide is, de szerencsére az Ubuntu-val már gyárilag használhatóak az alábbi utility-k mint például: a Git a verziókövetéshez, a bash amivel parancsokat futtathatunk és az apt is amivel különböző csomagokat töltehetünk és telepíthetünk az Ubuntunkra.

5.5 Windows

Régebbi dokumentációkat olvasva azt vehetjük észre, hogy hangsúlyozva van a megfelelő Windows verzió is, mivel a WSL 2 és Cuda kompatibilitás egy friss változásnak számít még. Szerencsére, ha a legfrissebb verzióval rendelkezünk, akkor a kompatibilitással nincsen probléma, ellenkező esetben minimum egy 21H2-es verziót telepítenünk kell, hogy a WSL ki tudja használni videókártyánk erejét.

5.6 Windows Power Shell

Ez az első teljesen opcionális program, amit csak kényelem szempontjából érdemes letölteni. Alapvetően egy alap Command Prompt megfelelne a céljainknak, de ez a Windows Store-ból

letölthető command line rendkívül hasznos tud lenni, mivel egyenesen meg tudjuk benne nyitni a WSL-t egy új tab-on. Emellett esztétikailag és funkciókban is gazdagabb.

5.7 NodeJs/NVM

Annak érdekében, hogy le tudjuk magunknak futtatni az adattisztítást vagy esetleg a későbbiekben egy szervert és webklienst a modell egyszerű kipróbálására, le kell töltenünk a NodeJs-t. Alapvetően a Javascript egy kizárólag böngészőben futó programozási nyelv, ám a széleskörű igény növekedése miatt, hogy fusson szerver oldalon is, megépítették a NodeJs-t. Most már a Node egy rendkívül elterjedt szerver megoldás, ami a Chrome V8 motorjára épült és előszeretettel használják a streaming szolgáltatók. Az NVM, vagy másnéven a Node Version Manager, egy olyan eszköz, ami nagyban le tudja egyszerűsíteni a NodeJs különböző verzióinak telepítését és azok közötti váltást. Navigáljunk az alábbi oldalra: https://github.com/coreybutler/nvm-windows/releases és töltsük le az nvm-setup.zip fájlt. Csomagoljuk ki, majd futtassuk a telepítőt. Ezek után nyissunk egy Windows Power Shell-t és írjuk be az alábbi parancsot.

```
nvm install 13.13.0
```

Várjuk meg a telepítés végét és ha rákérdez az npm telepítésére, akkor menjünk az igenre. Az npm, vagy más néven Node Package Manager, a Javascript alapú projektek dependenciájának telepítésért felelős és a későbbiekben szükségünk lesz rá. Ezek után gépeljük be az alábbi parancsot és kész.

```
nvm use 13.13.0
```

Három lépésben telepítettük a NodeJs-t a rendszerünkre.

6. Betanítás

Egy DeepSpeech modell betanításához sok hardware követelménynek szerencsére nem kell megfelelni, de időnk megtakarítása érdekében erősen ajánlott. A köztudattal ellentétben nem igényel hatalmas hardware beruházást, de videókártya hiányában a betanítás 100x-120x lassabban történik. Ezt sajnálatosan tapasztalatból írom, hiszen az első betanításom során a Tensorflow egy driver kompatibilitási hiba miatt nem használta ki a videókártyám által kínált erőt és csak is kizárólag a processzorra hagyatkozott. A szakdolgozat írásakor a számítógépem az alábbi komponenseket tartalmazza: GTX 1070 videókártya, intel i7 8750-h processzor, 16GB RAM, amivel a körülbelül a 20 órányi hanganyagot 8-10 óra alatt feldolgozta. A betanítás során azt figyeltem meg, hogy az algoritmus mind a processzort, mind a videókártyát rendesen

terheli, néha képes 90-100%-os kihasználtság mellett is. A memória használat elérheti a 11-13GB-ot is, de ez szerintem annak tehető be, hogy maximálisan kihasználja az elérhető mennyiséget és teljesen stabilan működne 8GB RAM mellett is.

6.1 Projekt felépítése

Indítsuk el a Windows Power Shell-t és a legördülő listából válasszuk ki az Ubuntu-t. Emellett nyissunk meg egy fájl böngészőt is és a felső sávba írjuk be az alábbi parancsot, hogy a WSL fájlrendszerébe lépjünk

```
\\wsl$
```

Ez nem egy szükséges lépés, de azoknak, akik először használnak Linux alapú operációs rendszert, megkönnyebbülés lehet a szokásos Windows fájlkezelővel navigálni a mappákban. Lépjünk bele a letöltött Ubuntu verziónkba, a Power Shell-ben pedig adjuk meg a default user nevét. Ezek után lépjünk át a gyökérkönyvtárba, abban az esetben, ha alapvetően nem oda mutatna a command line.

```
cd /
```

Klónozzuk le a DeepSpeech repository-t GitHub-ról az alábbi paranccsal:

```
git clone --branch v0.9.3 https://github.com/mozilla/DeepSpeech
```

6.2 Dependenciák telepítése

Python

Az első legfontosabb dependencia amit ellenőriznünk kell az a Python verziója. Írjuk be ezt a parancsot

```
python --version
```

Abban az esetben, ha nem 3.6-os verzióval rendelkezünk, telepítenünk kell azt, különben kompatibilitási problémákba fogunk ütközni az egyéb dependenciákkal. Kövessük az alábbi lépéseket a Python 3.6-os verziójának a telepítéséhez:

Az {x} jelöli a jelenlegi verziót.

```
sudo apt-get install software-properties-common

sudo add-apt-repository ppa:deadsnakes/ppa

sudo apt update

sudo apt install python3.6
```

```
sudo update-alternatives --install /usr/bin/python python
/usr/bin/python3.6 1

sudo update-alternatives --install /usr/bin/python python
/usr/bin/python3.{x} 2

sudo update-alternatives --config python
```

A felugró kérdésbe válasszuk ki a 3.6-os verziót. Ezek után ellenőrizzük, hogy sikeresen telepítette a 3.6-os pythont az alábbi parancs segítségével:

```
python --version
```

A fent említett parancsok letöltötték a szándékozott verziónkat, beállítottunk rá egy prioritást, majd az utolsó parancs és a válaszunk segítségével sikeresen alapvető verzióvá tettük a 3.6-os Python-t.

Cuda/cuDNN

Ha rendelkezünk videókártyával, akkor ellenőrizzük, hogy Cuda kompatibilis-e az alábbi oldalon: https://developer.nvidia.com/cuda-gpus. Abban az esetben, ha nem találjuk videókártyánkat a listában, akkor sajnálatosan nem tudjuk használni és ez a rész átugorható.

A DeepSpeech dokumentáció szerint a Python 3.6-os verziójához a Cuda 10.0 szükséges a számunkra. Töltsük le az alábbi oldalról a szoftvert, majd másoljuk be a fájlt a Wsl/Ubuntu/etc mappába, amit a Windows File Explorer segítségével is megtehetünk.: https://developer.nvidia.com/cuda-10.0-download-archive?target_os=Linux&target_arch=x86_64&target_distro=Ubuntu&target_version=1804&target_type=runfilelocal. Abban az esetben, ha engedélyek miatt kapunk hibaüzenetet, akkor használjuk a beépített Linux parancsokat a másolásra. Tegyük fel, hogy az asztalra töltöttük le. Ebben az esetben a parancs így fog kinézni, ahol a {user} a Windows-os felhasználónk neve.

```
sudo cp -i /mnt/c/Users/{user}/Desktop/cuda_10.0.130_410.48_linux.run /etc
```

Ezzel nem csak fájlt, de mappákat is tudunk másolni és vegyük figyelembe, hogy könnyedén elérjük WSL-n keresztül a Windows-os fájljainkat. Ahhoz, hogy futtathassuk, először be kell navigálnunk az etc mappába, amit az alábbi paranccsal tehetünk meg

```
cd /etc
```

Ezek után gépeljük be a futtató command-ot és települt is a Cuda 10.0 verziója.

```
sudo sh cuda_10.0.130_410.48_linux.run
```

Emellett szükségünk lesz a cuDNN 7.6-os verziójára, amihez sajnálatosan először regisztrálnunk kell az nvida developer programjára. A folyamat ingyenes és pár percet vesz igénybe, utána folytathatjuk a telepítést. Regisztráció után töltsük le az alábbi fájlt és az előzőhöz hasonló módon másoljuk be az etc mappába.: https://developer.nvidia.com/compute/machine-learning/cudnn/secure/v7.6.0.64/prod/10.0_20190516/Ubuntu18_04-x64/libcudnn7_7.6.0.64-1%2Bcuda10.0_amd64.deb. Navigáljunk be az etc mappába majd az alábbi paranccsal telepítsük a cuDNN könyvtárat:

```
sudo dpkg -i libcudnn7-dev_7.6.4.38-1+cuda10.0_amd64.deb
```

Ha más verzióra volna szükségünk, akkor az alábbi oldalon tudunk keresni: https://developer.nvidia.com/rdp/cudnn-archive.

6.3 Adattisztítás szüksége

A Common Voice adathangtár a dolgozat írásakor körülbelül 14-15000 validált hangmintával rendelkezik. Megfigyeléseim alapján azok az egyedek definiálhatóak validáltként, amelyek minimum 2 úgynevezett upvote-val és 1 vagy kevesebb downvote-val rendelkeznek. A forrás több tsv (tab separated value) fájlból és mp3 hangfájlokból áll. Ezek a dev.tsv, invalidated.tsv, other.tsv, reported.tsv, test.tsv, train.tsv, validated.tsv és az mp3 formátumú hanganyagok. A tsv fájlok tartalmazzák a hangdonorok azonosítóját, a hozzátartozó mp3 fájl nevét, a mondatot szövegként és egyéb adatokat, mint pl.: up_votes, down_votes, age, gender, accents, locale, segment. A mondatok tartalmaznak kis és nagybetűket, mondatvégi írásjeleket, kötőjelet, vesszőt és egyéb speciális karaktereket. A betanítás pontossága érdekében minden karaktert kis betűre kell átalakítani, a speciális karaktereket és mondatvégi jeleket el kell távolítani, mert a modell minden karakterhez egy bizonyos hangot próbál majd párosítani. Emellett a tsv fájlokat csv (comma separated values) fájlokká kell alakítani és tartalmazniuk kell a hozzátartozó hangfájlok nevét, azok méretét bytes-ban, és a transcriptet. Az átalakítást el lehet végezni excelben, ám számomra egyszerűbb, gyorsabb és újrahasznosíthatóbb megoldásnak számít egy egyszerű Javascript algoritmus. A hangfájlokat wav formátummá kell átalakítani, de ehhez használhatjuk a DeepSpeech beépített importer-ét.

6.4 Hangok importálásának előkészítése

Először is hozzunk létre a DeepSpeech mappában egy alphabet.txt fájlt, amire mostantól repository-ként fogok hivatkozni. Itt fogjuk végezni a betanítást és egyéb command-ok megadását. Az alphabet.txt-t úgy építsük fel, hogy minden sora csak egy karaktert tartalmazzon,

ebbe beleértve a szóközt is, aminek külön sort adunk mivel a csend is egy fajta hangnak számít az algoritmus szemében. Ennek a fájlnak az a lényege, hogy importálásnál csak olyan szövegeket fog beimportálni, amit mi megadunk a fájlban és futási időben megszakítja hibával a betanítást, ha olyan karakterrel találkozik, ami nincs feltűntetve az alphabet.txt-ben. Nekünk az utóbbi része nagyon fontos, mivel nem szeretnénk, hogy az algoritmus speciális karakterekhez is hangot párosítson. Ez egy felesleges lépésnek tűnhet, mert előre kitöröljük a már várt speciális karaktereket, de a jövőben, ahogy nő a common voice adatbázisa, úgy nő vele az esély, hogy beleütközzünk egy új, eddig ismeretlen karakterbe, ami miatt megállhat a betanítás. Tűntessük fel ide a magyar abc, vagy az általunk választott abc összes kisbetűjét és egy szóközt. Az utolsó sort hagyjuk üresen szóköz nélkül. Ezt a dokumentáció részletezte. A magyar abc-t tartalmazó alphabet.txt megtalálható a GitHub repository-mban.: https://github.com/AdamK18/szakdolgozat

A beimportálás alapvetően egy kihagyható lépés lenne, hiszen mi magunk fogjuk átalakítani a tsv fájlokat csv fájlokká és határozzuk meg az adatok megoszlását, ám nincsen olyan gyors és egyszerű megoldásunk az mp3 fájlok wav fájlokká konvertálásában, mint a DeepSpeech-nek. Ellenkező esetben nyugodtan átugorható lépés lesz.

Ha létrehoztuk az alphabet.txt-t a repository-ban, akkor ezt másoljuk át a „data” mappába is és írjük felül az egyezést. Erre azért van szükség, mert a data mappában található txt felel a futási időben való ellenőrzésért. Az importálást az adattisztítás után fogjuk elvégezni.

A validated.tsv tartalmazza az összes felhasználható egyed adatát. A train.tsv, dev.tsv és a test.tsv a validated.tsv tördelékei, amelyek az összes validált adat 50, 20 és 30 százalékáért felelnek. A tördelt fájlokkal is lehetne átalakítást végezni, ám az adatok arányai nem megfelelőek, ezért a tisztító algoritmus a validated.tsv fájlt olvassa be, majd új arányokat hoz létre: 85% tanító, 10% validáló és 5% teszt adat. Értelemszerűen a tanító adattal fejleszti az algoritmust, a teszt sorokkal egy utolsó végleges tesztet végez, míg a validáló adatokat minden iteráció végén leteszteli, hogy fejlődést értünk-e el vagy túltanítottuk-e.

6.5 Adattisztítás futtatása

Töltsünk le egy text editort, amivel egyszerűen módosíthatunk fájlokat, hogy konfigurálhassuk azokat céljaink legoptimálisabb elérése érdekében. A Visual Studio Code a legelterjedtebb kód szerkesztő a piacon, én is ezt ajánlom a projekt folytatására. Az alábbi linken letölthető: https://code.visualstudio.com/

A kicsomagolt Common Voice mappában megtalálható a validated.tsv nevű fájl. Ezt a mappát nyissuk meg egy új Visual Studio Code ablakban és hozzunk létre egy index.js nevű fájlt. Másoljuk be ide az algoritmus kódját, majd nyissunk egy terminált és írjuk be, hogy „node index.js”. Ez legenerál három csv fájlt, amire a későbbiekben szükségünk lesz.

6.6 Az algoritmus működése

```
const fs = require("fs");

fs.readFile("validated.tsv", "utf8", (err, data) => {
  const arr = data.split(/\r?\n/);
  const initial = "wav_filename,wav_filesize,transcript\n";
  let train = initial;
  let dev = initial;
  let test = initial;

  for (let i = 1; i < arr.length; i++) {
    const audio = arr[i].toLowerCase();
    if (audio === "") continue;
    const step = i % 20;
    const audioArr = audio.split("\t");
    const audioName = audioArr[1].replace("mp3", "wav");
    var size = fs.statSync(`clips/${audioName}`).size;
    let transcript = audioArr[2].replace(/[?.…,;!":’]/g, "").trim();
    transcript = transcript.replace(/[--]/g, " ");
    transcript = transcript.replace(/  /g, " ");
    const info = `${audioName},${size},${transcript}\n`;

    if (step <= 17) {
      train += info;
    } else if (step <= 19) {
      dev += info;
    } else {
      test += info;
    }
  }

  fs.writeFile("train.csv", train, "utf-8", () => console.log("finished"));
  fs.writeFile("dev.csv", dev, "utf-8", () => console.log("finished"));
  fs.writeFile("test.csv", test, "utf-8", () => console.log("finished"));
});
```

Ennek a sornak a segítségével beolvassuk a validált adatokat:

```
fs.readFile("validated.tsv", "utf8", (err, data) => {});
```

A tisztító algoritmust a kapcsos zárójelek közé kell írnunk, hogy megfelelően lefusson. Beolvasás után minden új sornál megtöri a szöveget és tömbként tárolja el őket egy változóban.

Létrehozzuk a train, dev, test szöveg alapú változókat, amelyek majd a későbbiekben az új fájljaink adataiként fognak szolgálni. Ezeknek adunk kezdeti értéket, pontosabban a DeepSpeech által megkövetelt fejlécet, ami így néz ki: „wav_filename,wav_filesize,transcript\n”. Láthatjuk a vesszővel való elválasztást és a végén az úgy nevezett escape charactert egy n betűvel párosítva, ami egy új sorért lesz felelős.
Egy for ciklussal végig iterálunk a beolvasott sorainkon és az alábbi modifikációkat hajtjuk végre:

- Kisbetűvé alakítás
- Speciális karakterek eltávolítása (? . , ; ! - –)
- Mondat eleji és végi szóközök és adattisztítás közben létrejött dupla szóközök eltávolítása.

Emellett a NodeJs által kínált „fs” package segítségével megvizsgáljuk a sorban megjelölt hangfájl méretét is. A hangfájl címét, méretét és leírását egy szöveggé kombináljuk vesszővel elválasztva és a végére rakunk egy sortörést escape character segítségével. A maradékos osztásra épült logika alapján eldöntjük, hogy az adott sornak tanító, validáló vagy teszt adatnak kell lennie és hozzáfűzzük a hozzátartozó változóhoz. Például, ha az index maradékos osztás 20 kevesebb mint 17, akkor tanító adatnak minősül. A ciklus végeztével létrehozzuk a csv fájlokat a változóink segítségével és kiírjuk őket a tsv fájlok mellé.

6.7 Adatok importálása

A leírt lépések után rendelkeznünk kell három csv fájlal és mellettük a „clips” mappában rengeteg mp3 hang fájlal. Hozzunk létre a wsl/DeepSpeech mappában egy „audio” nevű mappát és ide másoljuk be a Common Voice/hu mappa tartalmát. Most még szükségünk lesz az eredeti mappa struktúrára, hogy a hangok a „clips” mappában legyenek tárolva.
Lépjünk át egy Windows PowerShell-be és navigáljunk át a repository-ba. Másoljuk be az alábbi parancsot, esetleg módosítsuk, ha eltérés van a fájlok elérési útvonalában.

```
bin/import_cv2.py audio
```

Ezzel megindul az audio fájlok átkonvertálása és az algoritmus is létrehozza a saját csv fájljait, amiket beilleszt a „clips” mappába. Ha véget ért a folyamat, akkor láthatjuk, hogy a DeepSpeech minden mp3 fájlra generált egy wav fájlt is ugyanolyan néven. Az mp3 fájlokat nyugodtan kitörölhetjük, azokra nem lesz már szükségünk. Töröljük ki a clips mappában található három generált csv fájlt és másoljuk be a Node scriptünk által generált csv fájlokat.
Ezzel már majdnem készen állunk a betanításra. Telepítettünk dependenciákat, megtisztítottuk az adatainkat és el is helyeztük őket úgy, hogy a betanítás által hozzáférhető legyen.

6.8 Virtuális környezet létrehozása

Érdemes létrehozni egy Python virtuális környezetet az utolsó dependencia telepítéshez. Ez arra szolgál, hogy minden környezetnek a saját követelményei legyenek letelepítve. Például, ha két külön projekttel rendelkezünk, akkor ne globálisan kelljen a Tensorflow verzióját változtatni minden egyes váltásnál, hanem egyszerűen aktiváljuk a másik környezetet és máris hozzáférhetünk az ahhoz használt dependenciákhoz és verzióihoz. A virtuális környezet létrehozásához lépjünk át a repository-ba a Windows PowerShell-ben és írjuk be az alábbi parancsot:

```
python3 -m venv /DeepSpeech/env
```

Az aktiválásához gépeljük be az alábbi parancsot és látnunk is kell, hogy megváltozott a command line interfésze, ezzel jelezve, hogy átléptünk a virtuális környezetbe.

```
source env/bin/activate
```

Az előkészítés utolsó lépéseként még szükséges pár csomag telepítése. Ehhez kövessük a dokumentációt és gépeljük be az alábbi parancsokat. Ha nincsen letelepítve a pip3 package, akkor először ezeket a parancsokat másoljuk be:

```
sudo apt-get update

sudo apt-get -y install python3-pip

pip3 --version
```

Az utolsó körös dependenciák telepítéséhez:

```
pip3 install --upgrade pip==20.2.2 wheel==0.34.2 setuptools==49.6.0

pip3 install --upgrade -e .

sudo apt-get install python3-dev

pip3 uninstall tensorflow

pip3 install 'tensorflow-gpu==1.15.4'
```

Értelemszerűen a dolgozat írásának idejében érvényesek ezek a verzió számok. Ha jövőben változnának, akkor az alábbi oldalon találhatóak meg az aktuális követelmények: https://deepspeech.readthedocs.io/en/latest/TRAINING.html.

6.9 Betanítás indítása

Telepítettünk minden előkövetelményt és már csak egy lépés választ el minket a betanítás elindításától. Annak érdekében, hogy kényelmünkre megállíthassuk és folytathassuk, ahol abbahagytuk, meg kell adni egy külön paramétert majd a parancsnak és egy mappa elérési útját,

hogy elmentse a folyamatot. Bizonyosodjunk meg, hogy a repository-ban állunk, majd hozzunk létre egy mappát az alábbi paranccsal:

```
mkdir fine_tuning_checkpoints
```

Az algoritmus körülbelül 10-15 percenként menti el az állapotát a megadott mappába. Emellett hasonló módon meg kell adnunk az eredmény mappát is, amibe a tesztelés után a modell elmentésre kerül. Hozzunk létre még egy mappát az alábbi paranccsal:

```
mkdir output_models
```

Bizonyosodjunk meg róla, hogy a repository-ban állunk és aktiválva van a virtuális környezet. Ezek után másoljuk be az alábbi parancsot a betanítás elindításához:

```
python3 DeepSpeech.py --n_hidden 2048 --checkpoint_dir
fine_tuning_checkpoints/ --epochs 10 --train_files audio/clips/train.csv --
dev_files audio/clips/dev.csv --test_files audio/clips/test.csv --
learning_rate 0.0001 --export_dir output_models/ --use_allow_growth true --
train_cudnn true
```

Láthatjuk, hogy a parancsban megadunk különböző paramétereket, amik befolyásolják a betanítás kimenetelét. Tartalmazza a checkpoint és az output mappákat, a train/teszt/dev csv fájlokat és a „—train_cudnn true” parancsot is, ami videókártya kihasználásáért felel. Az algoritmus feltételezi, hogy a csv fájlok mellett tároljuk az audio fájlokat. A többi paraméter a dokumentáció alapján lett beírva.

Ha mindent pontosan végeztünk el, akkor el kell indulnia a folyamatnak és hallanunk kell ahogy számítógépünk elkezdi a munkát. A betanítás lépésekben és epoch-okban működik. Minden lépés egy audio feldolgozása, az epoch száma jelöli azt, hogy hány alkalommal tanította be az összes hangfájlt. A tanító fájlok feldolgozása után a rendszer leteszteli a dev adatokkal a modellt és minden epoch után és felméri a pontosságát. A legjobb pontosságú epoch-ot elmenti és amint eléri az epoch-ok száma a parancsban megadottat, akkor befejezi és a teszt adatokkal egy végső tesztet futtat rajtuk. Ezek után legenerálja az output mappába a modellünket egy pb fájlként.

Annak érdekében, hogy egy optimális modellt kapjunk, ahhoz át kell konvertálnunk a pb fájlunkat pbmm modellé, amit az alábbi parancs beírásával tehetünk meg:

```
sudo ./convert_graphdef_memmapped_format --
in_graph=output_models/output_graph.pb --out_graph
=output_models/output_graph.pbmm
```

Ezzel az output mappába generálja az optimalizált modellünket.

7. Tesztelés

A betanítás lefuttatása után a DeepSpeech megoszt velünk egy részletet az automatizált tesztek eredményéből. A Tensorflow több változóval is leírja a végső modellünk pontosságát. Ilyen például a WER – Word Error Rate, CER – Character Error Rate és a loss. A WER a szavak számának a pontosságát méri, vagyis, hogy a példa mondat x mennyiségű szóból áll és abból hányat talált el az algoritmus. A CER a karakterek pontosságát mutatja, hogy az összes karakterből hányat találtunk el pontosan. A loss egy általános értékelés, amely szintén a találgatás pontosságát méri fel. Mind a három esetben a kisebb érték számít jobbnak.

7.1 Az automatizált tesztek kiértékelése

Az eredményt kimásoltam a Windows PowerShell-ből és megvizsgáltam a mutatók értékeit. A betanításnál az én forrásmappám shared néven volt elmentve, ezért nem audio/clips van a helyén. Emellett konkrét példákat is mutat a legjobb WER eredményű tesztekről, a közepes pontosságúakról és a legrosszabbakról. A modell végeredménye:

Test on shared/test.csv - WER: 0.797104, CER: 0.255910, loss: 40.760906”.
Best WER:

WER: 0.000000, CER: 0.000000, loss: 4.293132

- wav: file://shared/common_voice_hu_25519461.wav

- src: "a hivatalos nyelvek a német és az alnémet"

- res: "a hivatalos nyelvek a német és az alnémet"

Median WER:

WER: 0.833333, CER: 0.137931, loss: 29.495985

- wav: file://shared/common_voice_hu_25537336.wav

- src: "ezen sorozat összegképletét többféleképpen is megkaphatjuk"

- res: "ezel sorozat összek képpletét tökféleképpen is megkapatuk"

Worst WER:

WER: 1.750000, CER: 0.227273, loss: 28.844313

- wav: file://shared/common_voice_hu_25347506.wav

- src: "lajos megbízottjaként portugáliában szolgált"

- res: "tajos megbízottja k ent portuk áliában falgát"

Mindegyik értékelésből kiragadtam egy-egy példát, hogy demonstráljam a modell pontosságát. Az eredmények első látszatra nagyon tetszettek, hiszen rögtön a legjobb példákkal szolgált először, de sajnos azok csak a szélső értékek. Közelebbi megfigyelések után feltűnik ugyanis, hogy az összesített modell eredmények inkább a közepes, vagy a rosszabb értékek felé hajlanak. Míg a Word Error Rate és Character Error Rate a medián tesztek pontosságához közelít addig a loss értéke bőven meghaladja a legrosszabb esetek pontosságát is. Összesen 15 ilyen példa mondatot gyártott le nekünk a DeepSpeech, minden kategóriába ötöt. A teljes fájl és az összes mondat megtalálható a Github repositorym-ban.: https://github.com/AdamK18/szakdolgozat

7.2 Előkészítés a manuális tesztelésre

Az automatizált tesztek eredménye egy részletes összegzést ad a modellünk működésének pontosságáról, ám nem hagyatkozhatunk egyedül azokra a számokra. A legjobb módja a teljesítmény felmérésére az, ha manuálisan leteszteljük mikrofonnal és egy felhasználói interfésszel. Ha követtük a leírásokat és telepítettük a Node 13-as verzióját, akkor könnyedén leklónozhatunk egy már működő prototípust és használhatjuk azt.

Klónozzuk le az alábbi Github repository-t: https://github.com/mozilla/DeepSpeech-examples. Ha ezzel is sikeresen megvagyunk, akkor nyissuk meg a repository „nodejs_wav” mappáját Visual Studio Code-ban.

Nyissunk egy integrált terminált a Terminal->New Terminal fülön majd írjuk be az alábbi parancsot:

```
npm install
```

A Node Package Manager felelős az összes dependencia telepítéséért és ezzel a paranccsal az összes ilyen csomagot letölti számunkra. Másoljuk be a mappánkba a frissen betanított modellünket, aminek pbmm formátumúnak kell lennie. Töröljük ki a „deepspeech-0.9.3-models.scorer” nevű fájlt, ha rendelkezik ezzel, mivel erre nem lesz szükségünk a továbbiakban. A scorer egy olyan fájl, ami tartalmaz különböző mondat kombinációkat. Ezzel képesek lennénk limitálni a modellünk szókincsét, mivel próbálna egyezéseket találni a scorer-ben talált mondatokból, de ugyanakkor képesek lennénk sikeresen növelni is a találgatás pontosságát. Sajnos egy ilyen fájl kiépítése egy bonyolult folyamat és nem éri meg ezzel a limitált betanító adatmennyiséggel.

A scorer kitörlése után nyissuk meg az index.js nevű fájlt. Annak érdekében, hogy a hangfelismerés működjön, a modellünk nevének meg kell egyeznie a kódban hivatkozott modellével. Ennek érdekében lehetőségünk van megváltoztatni a modellünk nevét „deepspeech-0.9.3-models.pbmm”-re vagy átírhatjuk a hivatkozott modell nevét a hatodik sorban. A legegyszerűbb, ha átírjuk a kódban hivatkozott nevet a saját modellünk nevére. A scorer-t kitöröltük, de a kódban még le van hivatkozva és próbálja is majd csatolni a modellhez. Ezért töröljük ki a tizennegyedik és tizenhatodik sort, amelyek a két fájl összehangolásáért felelősek. Emellett módosítsuk a tizennyolcadik sorban a wav fájl hivatkozását „audio.wav”-ra. Az előbb említett utasítások elvégzésével készen állunk arra, hogy rögzítsünk mikrofonunkkal szöveget és egyszerűen teszteljük azt ennek a script-nek a segítségével.

Utolsó lépésként telepítsünk vagy használjunk egy programot, amivel felvehetjük a tesztelni kívánt hanganyagokat. Az ingyenes Audacity programot ajánlom az egyszerű felvétel és wav formátum exportálás érdekében. A rögzítéshez használjuk a beépített mikrofonunkat vagy vásároljunk egy dedikáltat a pontosabb felvétel érdekében. Az Audacity-t az alábbi linken tudjuk letölteni: https://www.audacityteam.org/download/.

7.3 Manuális tesztelés

Az Audacity megnyitásával a felső sávban nyomjunk rá a nagy piros rögzítés gombra és beszéljünk bele a mikrofonba. A felvétel leállítása után Windows-os rendszeren a Ctrl+Shift+E kombinációval tudjuk gyorsan exportálni a hanganyagot. Ellenkező esetben a File/Export/Export As Wav opciót is használhatjuk. Mentsük el a hangot a „nodejs_wav” repository-ba audio.wav néven majd nyissunk egy Windows PowerShell-t a repository-ban. Gépeljük be az alábbi parancsot és pár másodpercen belül meg is jelenik a hangfelismerés eredménye.

```
npm start
```

Ezt a folyamatot lehet automatizálni bármilyen webes vagy egyéb felület segítségével, ami képes a rögzített hanganyagot egyenesen egy futó szervernek táplálni valamilyen API segítségével. Az API, vagy Application Programming Interface, a szerver olyan modulja, ami a külső kommunikációért felelős. Futtassuk le az általunk megtervezett teszteket, amely tartalmazzon különböző hosszúságú és komplexitású mondatokat. A pontosabb teszt érdekében különböző nemekkel és korosztályokkal is tesztelhetünk a diverzifikáció és lefedettség céljával.

8. Értékelés

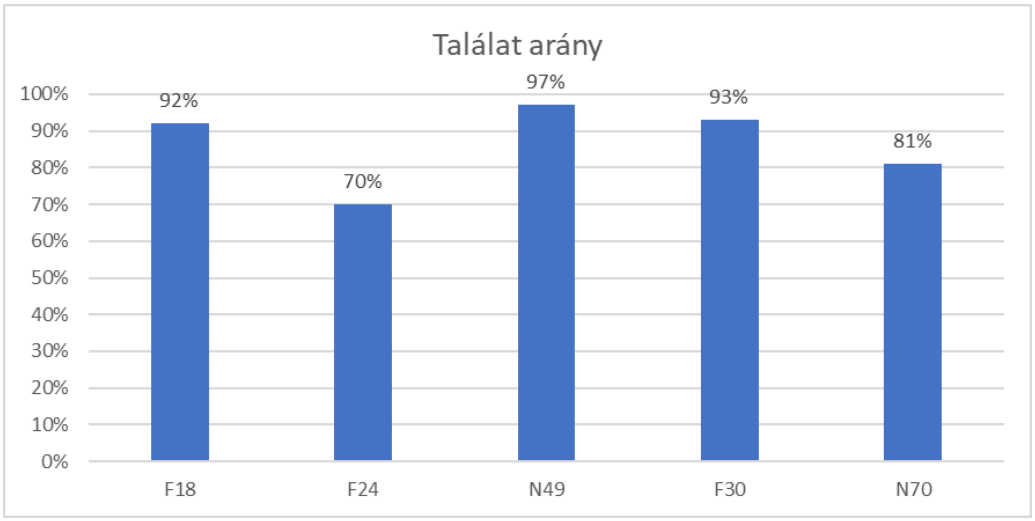
Az általam lefuttatott tesztek 20 darab mondatot tartalmaznak, egy-egy női és férfi megfelelőt. A példa mondatok és azoknak az eredményei az alábbiakban olvashatók.:

1. Megettem egy almát mert éhes voltam
   F: magattomagy almat metéhe sovoltan
   N: aremtev egy albát mezkéhezso voltom
2. Tanulnom kell a szerdán lévő vizsgámra
   F: tavólom ál a szebbel vévű isgemranni
   N: tanolnom kála aszerdál lébül ésmábol
3. A modell betanítása rendkívül nehéz volt
   F: a bodál betaitása vent tívül lehéz volt
   N: a odelbezalítá asap fen kimüllehéz olt
4. Elmegyek megsétáltatni a kutyámat és szívok egy kis friss levegőt
   F: elmagyek megsételtott nia kucsába tott és sziróbet kis hislelegőtu
   N: emegyék megsétáta nyi ak kösávan és szí ong egy kisukéslevedünt
5. Szabadidőmben szívesen játszok a számítógépen
   F: szabadidőmben széveséhátzokazamítógépen
   N: szavlidőnben sívásályjászok a szamító géban
6. Tavasszal utaztunk először külföldre
   F: tavatszaludasztónk előszertül fölül
   N: tavaszakotasztók előszer külfölde
7. A kék farmerhez fehért felsőt és fekete kabátot szeretnék felvenni
   F: a bék varmá hezefanyőnfelesüld és meketekabátótcelatnék felőönmén
   N: a kik varmárez fejévelsőd és lekete kavátont szeretlék lelelli
8. Ma négyig dolgozom de utána találkozhatunk
   F: madégyig dolkozo bótál takozhatómk
   N: a négyig dolgozom tótánatalákoszhatónk
9. Nagyon szeretnénk egy kis kutyát tartani
   F: agyon zeletlényké kis kucysáttaltak mr
   N: nagyon szeretnék egy kiskutyátartani
10. Általában csütörtökön járok ki a haverokkal inni
    F: általában külöltöken jerom ja havarokkalimni
    N: átolában csiterteköniáro kér averomkal inli

11. Szeretném ha erkély is tartozna a lakáshoz
    F: szeletlén a harkéis tartóznalakáshozbk
    N: szetném ha erké és tartoznalakácsoloz
12. Remélem hamarosan a diplomával díszeleghetek
    F: revélem havarasaldi plomábal bizleketek
    N: hemélem alarosal adipmomáladi szelethetek
13. A szomszédom tehetségesen utánoz egy galambot éneklés közben
    F: a szomszélottáhetségesen utánoz agy galanbopileplisközeben
    N: a szonszélonteségesen utánoz régolambond éleklés gözban
14. Szívesen megtanulnék tortát sütni és egyedül befalni
    F: szévesen megtadu légtoltá sükni és egyedünbefalnitak
    N: szilesen megtalurnét tóntátsog vé és kegyedübe mani
15. Hosszasan vártam a megállóban mert késve érkezett a villamos
    F: osszasan váltama megálló ban meg késlelyekezett a villamosonk
    N: a szesan vártam amegállóban megy késmegérkezett avillamos
16. A nagy takarítás előtt még el kell mennem a boltba
    F: a naystakaritásalött még álkál men nama boldva
    N: a negytakaritás elettúmég elkemelnem a boltba
17. Mindig kikötődik a cipőfűzőm az egyik lábamon
    F: mindik kitötüdikacípő füzőn az egyik lábón
    N: vmindig kiketődik a cipülfirzen az egyik lábamom
18. Nem tudom mennyi mindent csomagoljak el az útra
    F: lem tudom megnyiminden csonabola kal azóta
    N: em tudom megnyi minden csolakol jak el az ódra
19. A weboldalra feltöltött képeknek jó minőségűnek kell lennie
    F: a veboldolrak feltöltött képet neg jobmiősibülekellannia
    N: a vepoldara feltőltöt kéteklekjó midősígületellemlée
20. Sikerült időpontot foglalnom a fodrászomhoz
    F: sikelült időpontott foglalamafordlálszan oz
    N: siterült idő purtott foglallam a fodrászanoz

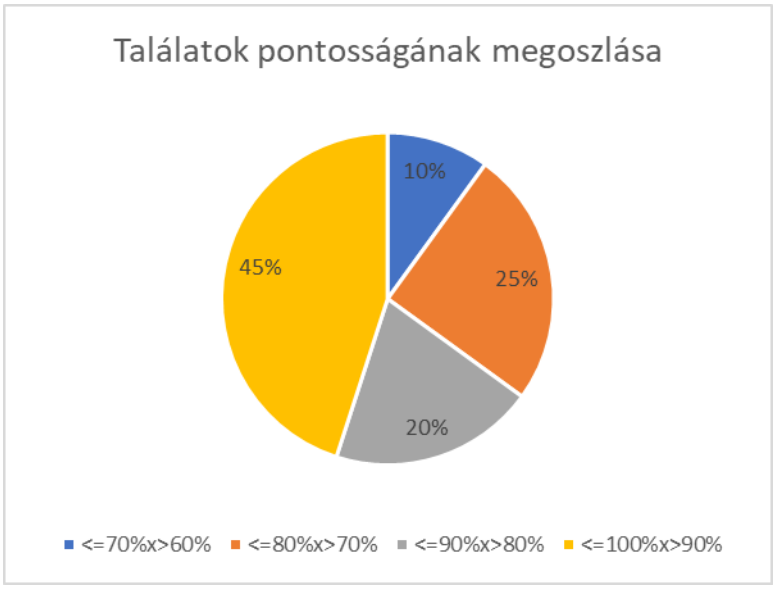
Az eredmények ránézésre sem hasonlítanak rendesen a magyar nyelvre. Egy két helyen kiszűrhető egy-egy helyes szó, de összességében egy átlagos embernek nehezére esne ezt elolvasni és értelmezni. Az értékeléshez más fajta megközelítés szükséges. A tesztelés után

kiválasztottam 5 különböző korosztályú embert. Egy 18 éves férfit, egy 24 éves férfit, egy 30 éves férfit, egy 49 éves nőt és egy 70 éves nőt. Az értékelés érdekében próbáltam velük kitaláltatni az eredeti mondatot a két generált mondat segítségével. A válaszukat az alábbi módon pontoztam. Telitalálat vagy mondat arányosan 1 apró hibázás 5 pontot érdemel. Az üresen hagyott válasz egy pontot kap. 20 százalékos hibázásonként a válasz elveszít 1 pontot. A válaszokat egy txt fájlba rendeztem és azok alapján excel-ben hoztam létre egy táblázatot. Szintén excel segítségével elkészítettem az alábbi hisztogramot az összesített eredményekről:



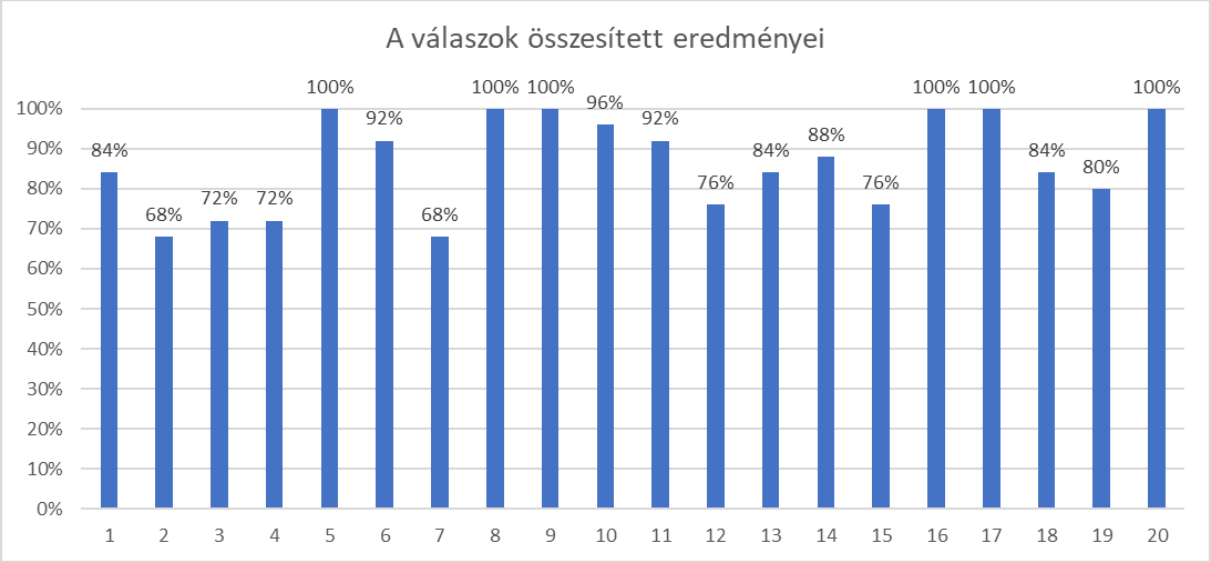
5. ábra Találat arány - Forrás: Saját készítés (2022)

A találati arány összesen **87 százalék**. Hatszor találták el mind az öten 100 százalékos sikerrel az eredeti mondatot és kettő alkalommal lett 68 százalékos, ami a leggyengébb eredmény. A találatok pontosságának a megoszlását az alábbi kör diagrammal szeretném bemutatni:



6. ábra: Találatok pontosságának megoszlása - Forrás: Saját készítés (2022)

Az utolsó ábra a válaszokkal kapcsolatos. Egy hisztogrammal szeretném ábrázolni, hogy milyen százalékos pontossággal tudták kitalálni az alanyok az egyes válaszok forrását:



7. ábra: A válaszok összesített eredményei - Forrás: Saját készítés (2022)

Ezekkel a számokkal egy kevésbé negatív szemmel nézhetünk rá a tesztek eredményére, de összességében még így is messze állnak a megfelelő értékektől.

9. Összegzés

Áttekintésképp, a dolgozatban szó volt a hangfelismerő rendszerek egyéni fejlesztési lehetőségeinek hiányáról és arról, hogy a DeepSpeech egy megoldást tudna nyújtani erre a problémára. Röviden bemutatásra került a mesterséges intelligencia fogalma, kialakulása, alkalmazásának területei és a gépi tanulás is. Emellett a hangfelismerésről, a történelméről, a működéséről és céljairól is szó volt. Említésre került a DeepSpeech és a Common Voice is, mint a dolgozat két fő pontja.

A betanítást a dependenciák telepítésével kezdtük, a fejlesztői környezet és a projekt felépítésével, majd a betanítás futtatásával. Az automatizált tesztek megvizsgálásával rájöttünk, hogy az adatmennyiség még nem megfelelő egy optimális modell kialakítására. Ennek ellenére a tesztalanyaink **87 százalékos** sikerrátával találták ki manuális tesztjeink eredményeit.

A DeepSpeech betanítása egy technikai műveletekben kevésbé jártas embernek elképzelhetetlen erőfeszítés volna. Még hozzáértéssel is gyakran voltak elakadások, félreértések és újrakezdések. Először a Docker virtualizált konténer környezetében próbáltam megoldani, de később kiderült, hogy a CuDNN nem támogatott a mai verziókon. A jövőben ez

valószínűleg változik, akkor ezzel a Docker Image-vel pár gombnyomással lehet telepíteni a dependenciákat: https://hub.docker.com/r/mozilla/deepspeech-train. Emellett a WSL/Ubuntu és a Python kezelése okozott még nehézséget.

Összességében a DeepSpeech és a Common Voice rendkívül hasznos kezdeményezések, amelyek 100 százalékosan a támogatásunkat érdemlik. A betanított modell is lehetett volna pontosabb, abban az esetben, ha több adathoz van hozzáférésünk. Az ajánlott mennyiségű adat több ezer óra. Ehhez képest a modell nem teljesített rosszul, hiszen körül belül a **2 százalékát** teszi ki. Ha azt szeretnénk, hogy a jövőben több kezdeményezés legyen magyar hangfelismeréssel kapcsolatban, ez esetben lehetőségünk van hozzájárulni ehhez a célhoz a Common Voice-on keresztül.

Irodalomjegyzék

Bansal, S. (2018): Turing Test in Artificial Intelligence, https://www.geeksforgeeks.org/turing-test-artificial-intelligence/, Letöltés dátuma: 2021.04.18

BBC (2014): Computer AI passes Turing test in 'world first', 2014 Június https://www.bbc.com/news/technology-27762088, Letöltés dátuma: 2021.04.18

CMN (2022): Common Voice, https://commonvoice.mozilla.org/hu, Letöltés dátuma: 2022.04.18

DPSCH (2021): Welcome to DeepSpeech's documentation!, https://deepspeech.readthedocs.io, Letöltés dátuma: 2021.04.18

HDSW (2021): How Does Speech Recognition Technology Work?, https://summalinguae.com/language-technology/how-does-speech-recognition-technology-work/, Letöltés dátuma: 2021.04.18

Microsoft (2021a): Mit jelent a mesterséges intelligencia?, https://azure.microsoft.com/hu-hu/overview/what-is-artificial-intelligence, Letöltés dátuma: 2021.04.18

Microsoft (2021b): Mi a gépi tanulás?, https://azure.microsoft.com/hu-hu/overview/what-is-machine-learning-platform/, Letöltés dátuma: 2021.04.18

SHOSR (2022): A short history of speech recognition, https://sonix.ai/history-of-speech-recognition, Letöltés dátuma: 2022.04.18

SUR (2019): Supervised Learning vs Unsupervised Learning vs Reinforcement Learning, https://intellipaat.com/blog/supervised-learning-vs-unsupervised-learning-vs-reinforcement-learning/, Letöltés dátuma: 2021.04.18

Turing, A. (1950): Computing Machinery and Intelligence, Mind, Vol. 59. No. 236., pp. 433-460., https://academic.oup.com/mind/article/LIX/236/433/986238

WAI (2021): What is Artificial Intelligence (AI)?, https://builtin.com/artificial-intelligence, Letöltés dátuma: 2021.04.18