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

Biometrikus felhasználó azonosítás

Diplomaterv

Készítette
Boér Lehel

Konzulens
Dr. Zainkó Csaba

2019. december 12.

Tartalomjegyzék

Kivonat i

Abstract ii

1. Bevezetés 1
1.1. Biometrikus azonosítás . . . 2
1.2. Biometrikus rendszerek teljesítménye . . . 3
1.2.1. Equal Error Rate . . . 3
1.3. Beszélőfelismerés . . . 5
1.4. Az automatikus beszélőfelismerés története . . . 6
1.4.1. Jellemző kinyerés . . . 7
1.4.2. Jellemző normalizálás . . . 7
1.4.3. A beszélőfelismerő modellek története . . . 7
1.5. Korábbi eredmények . . . 7

2. Beszédadatbázisok 9
2.1. Beszédadatbázisok . . . 9
2.1.1. TIMIT . . . 9
2.1.2. CMU Arctic . . . 10
2.1.3. LibriSpeech . . . 11
2.1.4. VoxCeleb1 . . . 13
2.1.5. VoxCeleb2 . . . 14

3. Neurális hálózat alapú beszélőfelismerő rendszerek 15
3.1. Modern beszélőfelismerés . . . 15
3.1.1. Deep Speaker: an End-to-End Neural Speaker Embedding System . . . 15
3.1.2. Speaker Recognition from Raw Waveform with SincNet . . . 17
3.1.3. VoxCeleb2: Deep Speaker Recognition . . . 20
3.1.4. Utterance-level Aggregation for Speaker Recognition in the Wild . . . 21
3.2. A hangminták hossza . . . 22
3.3. Mérési környezet . . . 22
3.3.1. Adatok előfeldolgozása . . . 23
3.4. WaveNet classifier . . . 23
3.4.1. WaveNet . . . 23
3.4.1.1. Nyújtott kauzális konvolúció . . . 23
3.4.1.2. SoftMax eloszlás . . . 25
3.4.1.3. Reziduális blokkok . . . 25
3.4.2. Módosított WaveNet architektúra . . . 26
3.4.3. Eredmények . . . 27
3.5. Voicemap . . . 28
3.5.1. A projekt általános felépítése . . . 28

3.5.2. Az implementált modellek . . . 29
3.5.3. Beszédadatbázisok és generátorok . . . 31
3.5.4. Tanítás . . . 31
3.5.5. Kísérletek és optimalizálás . . . 32
3.5.6. Saját kísérlet: Triplet loss . . . 36
3.5.6.1. Voicemap implementáció . . . 37

4. Meta learning **40**
4.1. Few-shot learning . . . 40
4.2. Metrikus metatanítás . . . 41
4.2.1. Konvolúciós sziámi hálózatok . . . 41
4.3. Optimizációs metatanítás . . . 43
4.3.1. Optimizáló algoritmus modellezése . . . 43
4.3.2. Model-Agnostic Meta Learning (MAML) . . . 44
4.3.3. Reptile . . . 45
4.4. Modell alapú metatanítás . . . 46

5. Android alkalmazás beszélőfelismerésre **47**
5.1. Modell predikció a felhőben vagy lokálisan? . . . 47
5.2. Alkalmazás architektúra és általános működés . . . 48
5.2.1. Biztonsági funkciók . . . 49
5.2.2. Vektorok átlagolása . . . 51
5.2.3. Konfiguráció . . . 51
5.3. Implementáció . . . 51
5.3.1. Kliens alkalmazás . . . 51
5.3.1.1. Projekt felépítése . . . 52
5.3.1.2. Engedélyek . . . 52
5.3.1.3. Tárhely . . . 52
5.3.1.4. Felhasználói felület . . . 53
5.3.1.5. Osztálydiagram és részletes működés . . . 54
5.3.2. Szerver . . . 55
5.3.2.1. Flask . . . 55
5.3.2.2. Projekt felépítése . . . 56
5.3.2.3. Részletes működés . . . 56
5.3.3. Fejlesztési környezet . . . 59
5.3.3.1. Git . . . 59
5.3.3.2. Amazon EC2 . . . 59
5.3.3.3. cmder . . . 60
5.3.3.4. Anaconda . . . 60
5.3.3.5. JetBrains PyCharm . . . 60
5.3.3.6. Android Studio . . . 61
5.3.3.7. Genymotion . . . 61
5.4. Tesztelés és kiértékelés . . . 62
5.5. Használati esetek és továbbfejlesztési lehetőségek . . . 62

Köszönetnyilvánítás **63**

Irodalomjegyzék **64**

HALLGATÓI NYILATKOZAT

Alulírott *Boér Lehel*, szigorló hallgató kijelentem, hogy ezt a diplomatervet 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, 2019. december 12.

Boér Lehel
hallgató

Kivonat

A biometrikus azonosításra használható modern, neurális hálózat alapú beszélőfelismerés jelenleg széles körben kutatott, gyorsan fejlődő terület. Jelenleg nincs átfogó rendszerezés, összehasonlítás a rendszerek között és ingyenesen elérhető beszélőfelismerő alkalmazás sem különböző modellek teljesítményének összeméréshez.

Dolgozatomban bemutatom a korai beszélőfelismerő rendszereket, a mai modern neurális hálózat alapú megközelítéseket és az ingyenesen elérhető beszédadatbázisokat. Ismertetem a metatanítás fogalmát a hozzá kapcsolódó optimizációs technikákkal. Részletesen bemutatok két beszélőfelismerő modellt; egy módosított WaveNet architektúrát és egy sziámi konvolúciós hálózatot, amelyeken méréseket is végzek. Végül leírom az általam készített Android alapú beszélőfelismerő alkalmazás implementációs részleteit, optimizálását és a felhasználási lehetőségeit.

Abstract

Modern neural network-based speech recognition for biometric identification is a widely researched, rapidly evolving field. There is currently no comprehensive systematisation, system comparison, and free speech recognition application to compare performance across different models.

In my thesis I introduce early speech recognition systems, today's modern neural network-based approaches and free speech recognition datasets. I introduce the concept of meta-learning with the related optimization techniques. I present in detail two models of speech recognition; a modified WaveNet architecture and a Siamese convolutional network, on which I also make measurements. Finally, I will describe the implementation details, optimization, and applications of my Android-based speech recognition application.

1. fejezet

Bevezetés

Kontextus. Biometrikus felhasználó azonosítás során a felhasználót saját biológiai jellemzői alapján azonosítjuk. A biometrikus azonosító rendszer a felhasználók biometrikus adatait tárolja és ezek segítségével azonosítja őket. Ezen módszer széleskörű megjelenése a mobiltelefonok fejlődésének köszönhető. Az újabb okostelefonok már képesek ujjlenyomat, arc és hang alapú azonosításra is.

A biometrikus azonosításnak számos előnye van a mai hagyományos jelszó alapú bejelentkezéssel szemben. Jelszavakat elveszíthetünk, eltulajdoníthatják tőlünk, illetve ha valaki hozzáfér az email fiókunkhoz már a kétfaktoros jelszavas autentikáció sem biztosít elegendő védelmet [1]. A biometrikus jellemzők egyedik és fizikailag a felhasználóhoz tartoznak ezért nehéz megszemélyesíteni őket és jóval nagyobb védelmet biztosítanak a jelszavaknál. A nagyobb védelmen kívül gyorsabb és kényelmesebb használni őket, illetve a jelszavakkal ellentétben nem kell emlékezzünk rájuk.

Probléma. A jelenleg elterjedt biometrikus azonosítási módszerek körében a biometrikus jel rögzítéséhez bonyolult és drága eszközökre van szükség (retinaszkenner, ujjlenyomat olvasó, digitális tábla aláíráshoz stb.).

A biometrikus azonosítási módszerek közül kevésbé elterjedt és jelenleg nagyon kutatott téma a hangalapú felhasználó azonosítás. Ennek előnye, hogy a hang rögzítéséhez nincs szükség drága vagy csak az újabb okostelefonokban található rendszerekre (retinaszkenner, ujjlenyomatolvasó), csupán egy mikrofonra és felhasználói szempontból is kényelmes.

Napról napra újabb módszerek, modellek jelennek meg, amelyek nincsenek egységesen rendszerezve, összehasonlítva. Továbbá nem létezik olyan szabad szoftver amellyel tetszőleges modellt használva szimulálhatunk egy beszélőfelismerő rendszert.

Megvalósítás. Dolgozatomban bemutatom a beszélőfelismerés fejlődését, megvizsgálom a mai modern neurális hálózat alapú beszélőfelismerő rendszereket, a rendelkezésre álló beszédadatbázisokat. Bemutatok részletesen két beszélőfelismerő modellt, majd meta tanítás alapú optimizációs technikákat. Végül ismertetem az általam implementált android prototípus alkalmazást hangalapú biometrikus azonosításhoz.

Cél. A hosszú távú cél egy olyan beszélőfelismerő rendszer létrehozása, amely segítségével összehasonlíthatjuk a különböző modellek teljesítményét, illetve egy általánosan felhasználható beszélőfelismerő alkalmazás létrehozása hangalapú autentikációhoz.

Kontribúciók. A dolgozatom a következő kontribúciókat tartalmazza:

- Módosított Wavenet architektúra teljesítményének mérése beszélőfelismerésre.
- Mérések a voicemap modellen: VoxCeleb adatbázis, triplet loss.
- Android prototípus alkalmazás beszélőfelismerésre.

1.1. Biometrikus azonosítás

A biometria az emberek fizikai jellemzőinek mérésével és elemzésével foglalkozik. Alkalmazását tekintve három területet különböztetünk meg [2]:

- Felhasználó ellenőrzés: Az azonosító rendszer a biometrikus adatot egy, korábban rögzített mintához hasonlítja. Ez alapján dönt, hogy a felhasználó hozzáférhet-e a kívánt erőforráshoz. Ilyen egy ujjlenyomat-olvasóval ellátott mobiltelefonon a képernyőzár feloldása. A felhasználó ellenőrzés arra ad választ, hogy az illető az-e akinek mondja magát.
- Felhasználó azonosítás: Az azonosító rendszer a biometrikus adatot több korábban vett mintához hasonlítja és arra ad választ, hogy ki a felhasználó; azaz beletartozike a korábban eltárolt biometrikus adatokból álló csoportba vagy nem. Ilyen lehet például egy ujjlenyomat-leolvasóval ellátott beléptetőrendszer cégek esetében.
- Duplikátum detektálás: Annak ellenőrzése, hogy egy felhasználó egynél többször szerepel-e egy adatbázisban. Csalások, például szociális támogatást többször igénylők kiszűrésére használják.

Az első biometrikus azonosítási eljárás az ujjlenyomatvételen alapuló személyiségazonosítás volt, amely a modern kriminalisztika világában terjedt el, de manapság már megtalálható okostelefonokban, biometrikus beléptetőrendszerekben is.

A biometrikus azonosítást az ún. biometrikus azonosító rendszer végzi el. A folyamat során a biometrikus azonosító rendszer mintát vesz az azonosítandó egyén egy vagy több előre meghatározott fizikai jellemzőjéről, és ezekről digitális lenyomatokat képez. Az első, regisztrációs fázisban a biometrikus minta lenyomatát a rendszer egy adatbázisban eltárolja, majd később az azonosítás során az aktuális mintát összeveti a korábban rögzítettel és dönt az egyezésről. Ahhoz, hogy az ember egy fizikai jellemzőjét biometrikus adatként használhassuk, a következő elvárásokat támasztjuk vele szemben:

- Általánosság: A biometrikus adattal minden egyénnek rendelkeznie kell.
- Egyediség: A biometrikus adatnak egyedinek kell lennie a releváns populáción belül.
- Állandóság: A biometrikus adat nem, vagy csak keveset változzon az idő eltetével.
- Mérhetőség: Az biometrikus adat az egyén részéről legyen könnyen mérhető testi adottság.
- Teljesítmény: A biometrikus azonosító rendszerek teljesítménye: gyorsaság, pontosság, technológia.

- Elfogadottság: A releváns populáción belül a mérési eljárás mennyire elfogadott (emberi méltóság megőrzése).
- Biztonság: Mennyire nehéz utánozni, hamisítani a biometrikus adatot?

A biometrikus adat lehet fiziológiai (DNS, arc, ujjlenyomat, írisz) vagy viselkedési (hang, írás, gesztusok). Mivel ezek az adatok statisztikai jellegűek, megbízhatóságuk változó. Minél több adat van egy mintában, annál egyedibb, és minél nagyobb a releváns populáció (eltárolt minták összessége), annál valószínűbb, hogy találunk két hasonló mintát. Ennek elkerülésére manapság terjednek a multimódusú biometrikus azonosító rendszerek, amelyek több biometrikus adatot felhasználva végzik ez az ellenőrzés, azonosítás és duplikátum detektálás feladatát.

1.2. Biometrikus rendszerek teljesítménye

A beszélőfelismerő rendszerek teljesítményének mérésekor a fel nem ismert beszélőket és a rosszul felismert beszélőket vesszük figyelembe. Míg az előbbi kisebb probléma, hiszen a felhasználó újra próbálkozhat, egy illetéktelen felhasználó téves azonosítása jelentős biztonsági rés, ezért minimalizálnunk kell azt.

Egy beszélőfelismerő rendszernek kétfajta feladatot adhatunk [3]:

- célzott próba: Két hangmintát adunk a rendszernek egyazon beszélőtől.
- nem célzott próba: Két hangmintát adunk a rendszernek két különböző beszélőtől.

Mindkét esetben a rendszernek a feladata eldönteni, hogy a két hangminta ugyanattól a felhasználótól származik-e, vagyis megkülönböztetni a célzott és nem célzott próbákat egymástól. A felismerő rendszer mindkét próba esetén tévedhet, ez két féle hibát von maga után:

- hamis pozitív vagy téves riasztás: Egy nem célzott próbát célzottnak tekint.
- hamis negatív vagy téves visszautasítás: Egy célzott próbát nem célzottnak tekint.

A teljesítmény mérése több módszer is létezik. A NIST a *Detection Cost Function* (C_{det}) metrikát ajánlja, de ez alkalmazásfüggő. Ebben a fejezetben az EER mérőszámot mutatom be, mert kiszámolása jóval egyszerűbb és nem tartalmaz alkalamazásfüggő paramétereket.

1.2.1. Equal Error Rate

Az *Equal Error Rate (EER)* vagy *Crossover Rate (CER)* a biometrikus rendszerek teljesítményének mérésére használatos mérőszám. Az *EER* segítségével tudjuk eldönteni ellenőrzés vagy azonosítás esetén a döntési küszöbérték szintjét, amely mellett a rendszer legjobban teljesít, azaz így optimizáljuk azt. Például beszélőfelismerés esetén két hang különbségét valamekkora távolsággal jellemezzük. Ebben az esetben a küszöbérték alatt azt a távolságot értjük amely szintje alatt a két beszélőt azonosnak, fölötte különbözőnek tekintjük.

A biometrikus rendszereket három fontos mérőszámmal jellemezzük. Az FAR, az FRR és az előző kettőn alapuló EER. Az első kettőre általában angolul, a következőképpen hivatkozunk:

- False Acceptance Rate (FAR): A tévesen elfogadás esetén a biometrikus rendszer beenged egy illetéktelen felhasználót, azaz hiba miatt mással azonosítja őt. A téves elfogadási ráta ennek a mértéke, a téves elfogadások számának aránya az összes elfogadáshoz képest.

  $$FAR = \frac{FA}{AA},$$

  ahol FA a téves elfogadások száma, AA pedig az összes elfogadás száma.

- False Rejection Rate (FRR): A téves visszautasítás azt jelenti, hogy az illetékes felhasználót a rendszer hiba miatt nem tudja azonosítani. A téves visszautasítási ráta a téves visszautasítások számának aránya az összes visszautasításhoz képest.

  $$FRR = \frac{FA}{AA},$$

  ahol FR a téves visszautasítások száma, AR pedig az összes visszautasítás száma.

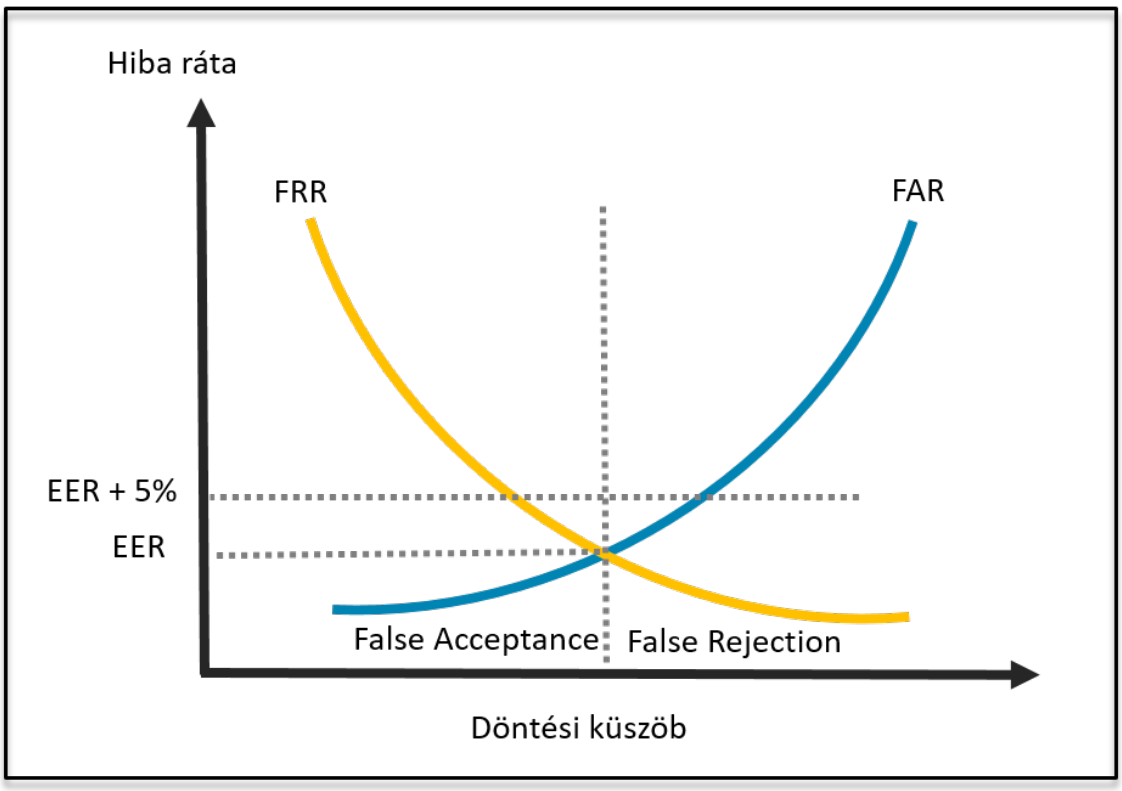
Az EER az a pont, ahol az FAR értéke megegyezik az FRR-el.

A biometrikus rendszerek karakterisztikáját a 1.1 ábra szemlélteti. Vegyük egy beszélőfelismerő rendszer példáját. Látható, hogy a döntési küszöb növelése (mekkora távolság esetén tekintjük azonosnak a két beszélőt) esetén az FAR nő és az FRR csökken. Ez normális, mert ilyenkor megengedőbbek vagyunk, nagyobb távolság esetén is azonosnak tekintjük a két beszélőt, így több olyan eset lesz amikor két hangminta különböző beszélőktől származik, a rendszer mégis azonosnak tekinti őket és kevesebb olyan amikor két hangminta azonos beszélőtől van és a rendszer mégis visszautasítja.

Ha a küszöböt csökkentjük szigorúbb lesz a rendszer, sokkal kevesebb téves elfogadás lesz, de a szigorúság miatt megnő a téves viszautasítások száma.

Az EER a két grafikon metszéspontjában található, ahol FAR és FRR értéke minimális és optimális (az optimizálás alatt érthetjük a kettő szorzatát). Az EER a vízszintes tengelyen megadja az optimális küszöbértéket, a függőlegesen pedig egy hiba rátát. Ez az FRR és FAR hiba rátája, mivel a kettő ebben a pontban megegyezik.

Az EER minimalizálásával tehát maximalizáljuk a biometrikus rendszer teljesítményét abból a szempontból, hogy a lehetséges hibák száma így a legkevesebb.



1.1. ábra. Biometrikus rendszerek karakterisztikája.

1.3. Beszélőfelismerés

Az emberi kommunikáció során fontos feladat a beszélő partner felismerése. A telekommunikációs technológia fejlődése miatt elterjedt a telefonon vagy interneten történő hangalapú kommunikáció; a telefonos felhasználófelismerés mint biometrikus azonosítási módszer megjelent már banki alkalmazásokban, call centerekben és az elektronikus kereskedelemben is (mobiltelefonos vásárlás). Az elektronikus kommunikáció során sokszor csak a beszélő hangjára hagyatkozhatunk, az alapján ismerhetjük fel az illetőt. A beszélőfelismerést háromféle módon végezhetjük [4]:

- Naiv beszélőfelismerés: Az emberi, naiv beszélőfelismerés során az ismerős hangokat meglepően nagy pontossággal ismerjük fel.
- Törvényszéki beszélőfelismerés: A törvényszéki szakértői vizsgálat eredménye.
- Automatikus beszélőfelismerés: A beszélőfelismerést számítógépes rendszer végzi.

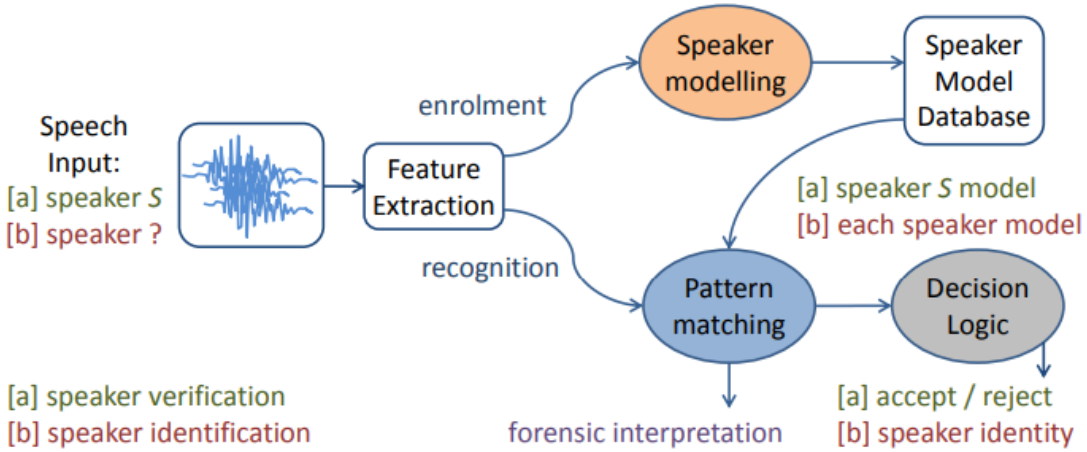
A beszélőfelismerés alatt három szűkebb fogalmat értünk. Ha a folyamat során az ismeretlen beszélőről azt ellenőrizzük, hogy az-e akinek állítja magát beszélő ellenőrzésről van szó. Beszélő szegmentáláskor a hangmintát homogén csoportokra bontjuk a beszélő személye alapján. Végül beszélő azonosításról beszélünk, ha az illető hangját rögzített hangok egy csoportjával vetjük össze és azt szeretnénk eldönteni, hogy melyikhez hasonlít a legjobban. Utóbbi felveti a kérdést, hogy mi történik ha a beszélő nem tagja a csoportnak.

Emiatt megkülönböztetjük a nyitott és zárt halmazú beszélőazonosítást. Utóbbi esetén csak olyan beszélőket ismerünk fel, akikről van hangminta az adatbázisban, míg az előbbinél ismeretlen beszélők is megjelenhetnek, így ezt is kezelni kell.

A beszélőfelismerés továbbá lehet szövegfüggő és szövegfüggetlen attól függően, hogy a felismerő rendszer egy előre meghatározott mondatot vár, vagy bármilyen hangminta alapján képes elvégezni a felismerést.

1.4. Az automatikus beszélőfelismerés története

Az automatikus beszélőfelismerést egy számítógépes program végzi emberi beavatkozás nélkül. Az első automatikus beszédfelismerő rendszert a Texas Instruments fejlesztetése volt és 1977-ben publikálták [5]. A rendszer szövegfüggő beszélőellenőrzésre volt képes és az évek során a téves elutasítási és elfogadási rátája 1% alatt maradt. A hetvenes évek óta a beszélőazonosító és ellenőrző rendszerek rengeteget fejlődtek kezdve a vektor kvantálástól a GMM modelleken át a mély neurális hálókig.



1.2. ábra. Korábbi automatikus beszélőfelismerő rendszerek architektúrája [6].

Az automatikus beszélőfelismerő általános működését a 1.2 ábra szemlélteti. Működését tekintve két fázist különböztetünk meg: A regisztrációs vagy felvételi fázisban hangmintát veszünk a felhasználótól [6]. Később az azonosítási vagy ellenőrzési fázisban a rendszer újabb mintát kér és az előbbi fázisban eltárolt hangmintával hasonlítja össze.

Mindkét fázis során az első lépés rögzített, alacsonyabb dimenziós *jellemzők kinyerése*, amelynek során a beszédet tömörebb formára hozzuk (pl. hangvektor). Az első, *regisztrációs fázisban* ezután statisztikai modelleket készít az egyes beszélőkhöz a kinyert jellemzők alapján, amelyeket eltárol a beszélő adatbázisban. Ebben a fázisban egy küszöbérték is meghatározásra kerül. *ellenőrzési fázisban* a rendszer kinyeri ugyanazokat a jellemzőket az aktuális hangmintából, majd egy *mintaillesztés* nevű eljárással összehasonlítja a korábban tárolt modellt a jellemzőkkel. Az összehasonlítás eredményét és a küszöbértéket figylembe véve a rendszer dönt az azonosítás vagy ellenőrzés eredményéről.

1. beszélő-ellenőrzés esetén megkeresi az ellenőrizendő személyhez tartozó modellt az adatbázisban és összehasonlítja az aktuális jellemzőkkel. Ha az eredmény a küszöbszint felett van, a rendszer egyezést mutat.

2. beszélőazonosítás esetén az aktuális jellemzőket az összes modellel összehasonlítja, majd a legjobb egyezés mellett dönt.

1.4.1. Jellemző kinyerés

A jellemző kinyerés célja a dimenzió csökkentése és a beszélőspecifikus információk kinyerése. Mivel a beszéd komplex jel, a beszélő azonosítása szempontjából felesleges információkat is hordoz. Ilyen például a környezet és a csatorna zaja. A kinyert jellemzőket a hangminta terjedelme alapján osztályozzuk. *Rövidtávú jellemzők* a 20-30 ms-os keretekből kinyert mel-frekvenciás és lineáris prediktív kepsztrális együtthatók (MFCC és LPCC) [4]. A *prozódikus jellemzők* kinyerése 100 ms-os terjedelemben történik és a beszéd ritmusát, a hangmagasságot és a sebességet jellemzik. A hosszútávú jellemzőket a jel akár perc hosszú kereteiből nyerjük ki. Ezek képesek reprezentálni a beszélő akcentusát illetve a szavak szemantikáját és az idiolektust.

A beszélőfelismerő rendszerek teljesítményét javította, ha a jellemzőket csak a hangminta azon részeiből nyerték ki, amikben beszéd is volt. Erre alkalmazott technika a *Voice Activation Detection* (*VAD*).

1.4.2. Jellemző normalizálás

Jellemzőkinyerés során próbáljuk kiszűrni a beszélő szempontjából értékes részéket, ugyanakkor nincs tökéletes jellemző, amely ne változna a környezet hatására. Ezt a változást segítik minimalizálni a normalizálási módszerek.

1.4.3. A beszélőfelismerő modellek története

Kezdetben a vektor kvantálás volt az elterjed modellezési módszer, amit később a *Gaussian Mixture Model* (*GMM*) váltott fel. A GMM egy adathalmazt több normális eloszlás keverékeként ír le és képes nem felügyelt módon klaszterezni az adatokat. Egy beszélőhöz egy valószínűségi sűrűségfüggvényt rendel, amely különböző pontokban kiértékelve (például teszt fázisban a beszélőtől kinyert jellemzők) egy valószínűséget ad a két beszélő hasonlóságára [4].

A GMM megközelítés főleg beszélőazonosításra alkalmas. Beszélő-ellenőrzéshez szükség volt egy másik modellre is, ami képes leírni minden más beszélőt az ellenőrizendőn kívül. Erre adott megoldást az *Universal Background Model* (*UBM*). Később jobb teljesítményt értek el, ha a teszt fázisban a beszélőkkel először UBM modelleket tanítottak és ezekből származtattak GMM-eket. Ezt nevezik GMM-UBM módszernek.

Mivel a tanító és teszt hangminták eltérő hosszúságúak lehetnek, szükség volt egy fix hosszúságú reprezentációra, ezt oldották meg a GMM szupervektorok, amelyeket az akkori megközelítés szerint szupport-vektor gépekkel vagy faktoranalízissel használtak.

Az utóbbi két módszer előnyeit kombinálva megszületett az $i-vektorok$, amelyet követve eljutunk a mai state-of-the-art módszerhez, a mély neurális hálózatokhoz (DNN).

1.5. Korábbi eredmények

A 1.1 ábra korábbi eredményeket mutat szövegfüggetlen beszélőfelismerő rendszerekről. Megfigyelhető, hogy főleg tízes, százas nagyságrendű beszélővel tanított, zárt-halmazú

rendszerekről van szó és a tanító mintákat is többnyire laboratóriumi körülmények között rögzítették. Ilyen környezetben könnyebb nagy pontosságot elérni, hiszen nincs háttérzaj, nincsenek új beszélők, a teszt minták nem feltétlenül elég diverzifikáltak. A pontosságot ennek fényében kell értelmezni.

Szerző (év)	**Szervezet**	**Populáció**	**Módszer**	**Jellemzők**	**Hang típusa**	**Pontosság**
Douglas A. Reynolds (1995)	Lincoln Laboratory	49	MFCC	rövid kifejezések	telefon	96.8 %
Rabah W. (2004)	King Abdulaziz University	20	SVD-alapú algoritmus	LPC/Cepstral	iroda	94 %
Yang Shao (2008)	Ohio State University	34	GFCCs	hallási jellemzők	telefon	~99.33 %
P. Krishnamoorthy (2011)	TIMIT	100	GMM-UBM	MFCC	labor	80 %
Alfredo Maesa (2012)	Voxforge.org	250	MFCC	spektrális jellemzők	beszéd-adatbázis	> 96 %
Sharada V. Chougule (2015)	Finolex Academy of Management and Technology	97	NDSF	spektrális	labor	~98-100 %

1.1. táblázat. Korábbi eredmények szövegfüggetlen beszélőazonosítás terén 2017 előtt [7].

2. fejezet

Beszédadatbázisok

A fejezet bemutatja a jelenleg ingyenes, bárki által elérhető és a tanítás során használt beszédadatbázisokat. Egy beszédadatbázis előállítása bár ránézésre egyszerű feladatnak tűnik, valójában komplex folyamat és sok munkát igényel. Egyes adatbázisokat beszédfelismerés céljából készítettek, másokat beszélőfelismerésre. Az előbbinél a kontrollált körülmények előállítása, megfelelés egyes nyelvtani mérőszámoknak (fonémák, diádok egyenletes elosztása stb.), utóbbinál a nagyobb méret miatt az automatizmus szükségessége bonyolítja a feladatot. Ezen felül mindkét esetben a szerzői jogok védelmére is ügyelni kell. (A dolgozatban a beszédadatbázis, beszédkorpusz és beszédadathalmaz ugyanarra vonatkozik.)

2.1. Beszédadatbázisok

2.1.1. TIMIT

A TIMIT beszédkorpuszt automatikus beszédfelismerő rendszerek fejlesztéséhez tervezték. 630 beszélőtől tartalmaz mintákat amerikai angol nyelven a 8 legelterjedtebb nyelvjárásban. A TIMIT archívum tartalmaz egy TRAIN és egy TEST mappát, ezek tanításhoz és teszteléshez valók. Ezeken belül további, a dialektusok sorszámával (DR1, ..., DR8), azon belül a beszélő azonosítójával elnevezett könyvtárak találhatók. Egy beszélőhöz 10 db beszédminta tartozik 16 kHz-es NIST SPHERE fájlok formájában [8].

Dialektus regió (DR)	**Férfi**	**Nő**	**Összesen**
1	31 (63%)	18 (27%)	49 (8%)
2	71 (70%)	31 (30%)	102 (16%)
3	79 (67%)	23 (23%)	102 (16%)
4	69 (69%)	31 (31%)	100 (16%)
5	62 (63%)	36 (37%)	98 (16%)
6	30 (65%)	16 (35%)	46 (7%)
7	74 (74%)	26 (26%)	100 (16%)
8	22 (67%)	11 (33%)	33 (5%)

2.1. táblázat. A beszélők eloszlása dialektusok szerint.

A dialektus régiók a következők:

- DR1: New England
- DR2: Northern
- DR3: North Midland
- DR4: South Midland
- DR5: Southern
- DR6: New York City
- DR7: Western
- DR8: Army Brat (moved around)

A NIST SPHERE formátum az elején definiál egy fix hosszú fejlécet, amit a hang bináris kódolása követ. Erre figyelni kell a hangfájlok beolvasásánál, Python esetében nem minden hangfeldolgozó könyvtár támogatja. Ilyen esetben kézzel el kell távolítani a fejlécet a fájlok elejéről. A hangfájlokhoz tartozik egy szöveges dokumentum ami az elhangzott szöveget és annak a wav fájlbeli helyét tartalmazza. Továbbá egy WRD fájl írja le a szavakat és egy PHN kiterjesztésű a fonémákat, illetve azok időbeni elhelyezkedését a wav fájlban.

A hangfájlokhoz tartozó egyéb fájlok beszédfelismerés szempontjából fontosak. Mivel én a beszédkorpuszt beszélőfelismerésre használtam, csak a hangfájlokra volt szükségem. A TEST mappában – mivel a TIMIT-et alapvetően beszédfelismeréshez tervezték – teljesen különböző beszélők vannak a TRAIN mappához képest, ezért a TRAIN mappabeli beszélőket osztottam fel tanításhoz és teszteléshez.

2.1.2. CMU Arctic

A CMU Arctic adatbázist beszédszintézishez kapcsolódó kutatásokhoz tervezték. A beszédszintézis az emberi beszéd mesterséges, általában számítógéppel történő előállítása. Mivel akkoriban (2003) a publikusan elérhető beszédadatbázisok kis méretűek voltak, létrehozták a CMU Arcticot, ami több beszédadatbázis összessége. Egy adatbázis egy beszélőhöz tartozik és egyéb információkat is tartalmaz, mint a fonetikus kiejtés vagy a beszédjel egyes jellemzőit (pitchmark) tartalmazó fájlok [9].

Mivel a CMU Arctic felhasználási célja a beszédszintézis, a felvételek studió minőségűek, mentesek külső zajoktól és a beszélő által keltett zajoktól is.

Az egyes Arctic adatbázisok közel 1150 kb. 1-4 másodperces beszédmintából állnak. Az egy beszélőhöz tartozó mintákat két azonos elemszámú A és B csoportra osztották, amelyek fonetikailag kiegyensúlyozottak. Ez azt jelenti, hogy az adott beszédmintában található fonémák körülbelül ugyanakkora gyakorisággal fordulnak elő mint az átlagos társalgás közben az adott nyelven. A beszédminták WAV fájlok formájában 32 kHz-en lettek mintavételezve. A felvételek öt beszélőtől tartalmaznak hangmintákat:

- férfi általános amerikai angol akcentus
- női általános amerikai angol akcentus
- férfi kanadai angol akcentus
- férfi skót angol akcentus
- férfi indiai angol akcentus

A CMU Arctic tartalmaz egy listát az egyes beszélők által felolvasott szövegekkel. Mivel ez egy mindenki által elérhető ingyenes beszédadatbázisnak készült és szabad szoftver licensszel rendelkezik, fontos szempont volt, hogy a felolvasott szövegek ne sértsenek szerzői jogokat. Emiatt a listán szereplő mondatok a legnagyobb ingyenes, szabadon felhasználható szövegkorpuszból, a Project Gutenbergből származnak. A Project Gutenberg célja szerzői jogokat nem sértő angol nyelvű könyvek gyűjtése és elérhetővé tétele.

A CMU Arctic eleinte 2,5 millió szóból és az azokat tartalmazó 168000 mondatból állt. Ezekből a FestVox beszédszintézis projekt segítségével kiválogattak 52000 olyan mondatot, amelyek kiejtése könnyű és hossza 5 és 15 szó közé esett. További szigorításként csak olyan szavak kerülhettek be, amelyek részei a CMUDICT lexikonnak, hogy elkerüljék az eltérő kiejtéseket.

Ezután a Festvox segítségével az 52000 mondatból kiválogatták a legnagyobb diád lefedettségű halmazt. A diád kettő, a triád három egymás utáni félhang együtteseként előálló nyelvi egység, amelyek lefedettségének mértéke a beszédszintézis során előállított beszéd minősége miatt fontos.

A kiválogatott részt kivéve egy újabb halmazt választottak ki, amelyek 668 és 629 mondatból álltak. Ezeket manuálisan átnézve tovább finomították.

Arctic	**Automatikus fázis**	**Kézi finomítás I.**	**Kézi finomítás II.**
A halmaz	668	597	593
B halmaz	629	541	539
Összesen	1297	1138	1132

2.2. táblázat. Arctic CMU automatikus és kézi fázisok.

A CMU Arctic diád lefedettsége 80% és az előbbihez hasonlóan triád lefedettsége 14%. Az előbbi megelőzi a TIMIT beszédkorpuszban lévő diád lefedettséget, utóbbiban viszont a TIMIT jobb a több (2343) mondat miatt.

Corpus	**Mondat**	**Szó**	**Fonéma**	**Fonéma lefedettség**	**Diád lefedettség**	**Triád lefedettség**
TIMIT-all	2342	20771	87819	100%	78,2%	19,4%
Arctic	1132	10045	39153	100%	79,6%	13,7%

2.3. táblázat. Arctic CMU vs. TIMIT.

2.1.3. LibriSpeech

2015-ben már vizsgálták a hangoskönyvek beszédszintézishez való használatát és léteztek ilyen célú beszédadatbázisok, de nem volt automatikus beszédfelismerés céljára létrehozott, ingyenes, mindenki által elérhető beszédkorpusz. A LibriSpeech beszédfelismerő rendszerek tanítására és kiértékelésére készült. Tartalma a LibriVox projektből származik, amelynek célja szabadon terjeszthető hangfelvételek készítése ingyenes, közkinccsé vált könyvekről, és ezek közzététele bárki számára az interneten [10].

A projekt körülbelül 1000 órányi beszédet tartalmaz 16 kHz-en mintavételezve. Tartalmaz ezek mellett több előre tanított beszédfelismerő modellt és Kaldi szkripteket beszédfelismerő rendszerek készítéséhez.

Az előfeldolgozási fázisban a könyvek szövegét normalizálták; kis és nagybetűk helyett csak nagybetűket használtak, eltüntették az írásjeleket és a gyakori rövidítéseket teljes szavakkal helyettesítették. Ezután a hangmintákat felismertették a Kaldi *gmm-decode-faster* dekóderével, amihez egy VoxCeleb adatbázison tanított modellt használtak.

Összemérték az előbbi modell által felismert szöveget az eredeti írással és azokat a részeket választották ki, ahol a kettő megegyezett. A hanganyagot legfeljebb 35 másodperces részekre tördelték minimum 0.5 másodperces szünetek mentén.

A következő fázisban egy fMMRl (HMM-alapú) modellel ismét összemérték a felismert szöveget a valódival. Ez okozott hamis pozitív eredményeket is, de az audioanyag mennyisége miatt elhanyagolható volt. Ez a fázis tehát 35 másodperces nagy valószínűséggel helyesen transzkriptált anyagokat tartalmazott.

A harmadik fázis ezeknek az apróbb szegmensekre tördelése volt, amit két módszerrel végeztek el. Tanítóadatok esetén minden olyan szünetnél tördeltek, ahol az legalább 0,3 másodperc hosszú volt. Tesztadatok esetén csak ott tördeltek, ahol ez a szünet egybeesett a mondat végével is.

Részhalmaz	**Óra**	**beszélő / perc**	**Férfi beszélő**	**Női beszélő**	**Összes beszélő**
dev-clean	5,4	8	20	20	40
test-clean	5,4	8	20	20	40
dev-other	5,3	10	16	17	33
test-other	5,1	10	17	16	33
train-clean-100	100,6	25	125	126	251
train-clean-360	363,6	25	439	482	921
train-other-500	496,7	30	564	602	1166

2.4. táblázat. LibriSpeech részhalmazok.

Az audiófelvételek válogatásához a LibriVox API segítségével szereztek információkat a felolvasóról, hogy mely hangoskönyvekben olvastak fel milyen fejezeteket, a kapcsolódó referencia szövegeket pedig a Project Gutenbergből és az Internet Archiveból nyerték ki. A *train*, *dev* és *test* részhalmazok között fontos, hogy ne legyen átfedés, tehát ugyanaz az olvasó csak egyetlen halmazban olvasson ezek közül. Ehhez a válogatáson kívül további óvintézkedésként kivették a többszereplős fejezeteket és lefuttatták a LIUM beszélő szegmentáló toolkitet ennek detektálására.

A tanítóhalmazba válogatott részeket három részre osztották. Ezek egyenként 100, 360 és 500 órányi audiofelvételt tartalmaznak. Az első kettőbe automatikus módszerrel az amerikai angol akcentushoz hasonló részeket válogatták bele. Egy egyszerű, a Wall Street Journal si-84 adatbázison tanított akusztikus modellt használtak a LibriSpeechen beszédfelismerésre. Ezáltal meghatározták és rangsorolták a beszélőket a Word Error

Rate (a modell által hibásan felismert szavak) szerint. Az alacsonyabb WER pontszámmal rendelkező olvasók a *clean*, a magasabbal az *other* részhalmazokba kerültek.

2.1.4. VoxCeleb1

A beszélőfelismerésre használt adatbázisok kontrollált körülmények között felvett audio anyagot tartalmaznak és kézzel válogatottak, emiatt kis méretűek. Kontrollált körülmény alatt értjük a studioban felvett, külső és belső zajoktól mentes hanganyagot. Ezek a valós körülmények közötti, azaz zajos környezetben elhangzó beszédfelismerést megnehezítik. A VoxCeleb készítői a beszélőfelismerő adabázisok automatikus létrehozására terveztek egy folyamatot és létrehozták, valamint ingyenesen elérhetővé tették a VoxCeleb beszédadatbázist, ami valós körülmények közötti hangfelvételeket tartalmaz [11].

Név	**Körülmények**	**Beszélők száma**	**Minták száma**
ELSDSR	Kontrollált	22	198
Forensic Comparison	Telefonos	552	1264
SITW	Multimédia	299	2800
VoxCeleb	Multimédia	1251	153516

2.5. táblázat. Beszédadatbázisok beszédfelismerés.

A VoxCeleb több mint 100000 olyan beszédmintát tartalmaz 1251 híresség̋től, amik YouTube videókból származnak. Az adatbázisban a nemek aránya kiegyensúlyozott (férfiak 55%, nők 45%), a beszélők különböző származásúak, korúak illetve eltérő akcentussal rendelkeznek. A beszéd általában publikus eseményekről származik, így a háttérzaj, egyéb beszélők és a felvevő eszköz minősége zajossá teszi azt.

Az adatbázis előállítása a következő módon történt:

- A beszélő jelöltek kiválasztása: Olyan híresség̋eket válogattak akik benne vannak a VGG adathalmazban, ami 2622 híres (interneten sokat keresett) ember arcát tartalmazza.
- YouTube videók letöltése: Letöltötték a kiválasztott emberekhez tartozó első 50 YouTube videót. A keresés az illető nevét és utána az *interview* szót tartalmazta, hogy kiszűrjék a zeneszámokat, sportfelvételeket és minél nagyobb valószínűséggel beszéljenek benne.
- Arcdetektálás: Az egyes videókon egy HOG-alapú arc detektorral meghatározták az arcok helyét minden egyes frameben.
- Aktív beszélőazonosítás: Mivel egy videóban több arc is megjelenhet egyszerre, vagy a beszélőt szinkronizálhatják, az éppen beszélő arc meghatározására illetve a szinkron kiszűrésére volt szükség. A megoldás a SyncNet volt, egy olyan kétcsatornás CNN, ami képes megbecsülni a szájmozgás és a beszéd közötti korrelációt.
- Arcfelismerés: Végül egy a VGG Face adathalmazzal tanított CNN segítségével ellenőrizték, hogy a beszélő arc valóban a kérdéses személy-e.

2.1.5. VoxCeleb2

A VoxCeleb2 ugyanúgy beszélőfelismeréshez készített, de az előző verzióhoz képest jóval nagyobb, 6112 híresség̋től tartalmaz 2442 órányi beszédet tartalmazó beszédadatbázis különböző nyelvű, korú és származású beszélőkkel [12].

Az adatbázis előállítási folyamatot az előzőhöz képest tovább bővítették.

- Duplikátum detektálás: YouTube-on gyakran feltöltik ugyanazt a videót más url alatt különböző felhasználók. Egy a VoxCeleb1 adatbázison tanított konvolúciós hálózatot használtak jellemző kinyerésre, amellyel 1024 dimenziós vektorokkal reprezentálták a beszédmintákat. Egy beszélőre kiszámolták a vektorpárok közötti Euklideszi-távolságot és ahol ez kisebb volt mint 0,1 azokat azonosnak tekintették.
- Nemzetiség: A beszélők nemzetiségét a Wikipediáról gyűjtötték ki egy keresőrobottal (angolul webcrawler). Ezzel a módszerrel a 6112 beszélő közül 5689 nemzetiségét sikeresen azonosították. Ebből kiderült, hogy a VoxCeleb2 etnikailag sokkal diverzifikáltabb a VoxCeleb1-hez képest, mivel a VoxCeleb1 adatbázisban jelenlévő 36 nemzetiségnél jóval többet, pontosan 145-öt tartalmaz. Ugyanakkor az amerikai angolt beszélők aránya is sokkal kisebb, 29% a VoxCeleb1-ben mért 64%-hoz képest.

Adatok	**VoxCeleb1**	**VoxCeleb2**
Beszélők száma	1251	6112
Beszélők száma	1251	6112
Női beszélők száma	561	2351
Férfi beszélők száma	690	3761
Videók száma	22496	150480
Beszédminták száma	153516	1128246
Átlagos videó / beszélő	18	25
Átlagos beszédminta / beszélő	116	185
Átlagos beszédminta hossz	8,2	7,8

2.6. táblázat. VoxCeleb1 vs. VoxCeleb2.

3. fejezet

Neurális hálózat alapú beszélőfelismerő rendszerek

Ebben a fejezetben bemutatom a legújabb korszerű, mély neurális hálózaton alapuló beszélőfelismerő rendszereket, majd ismertetem a WaveNet architektúrát és egy arra épülő beszélőfelismerő neurális hálózatot; a Wavenet classifiert. Végül bemutatom a voicemapet, egy beszélőfelismerő feladatokra használható könyvtárat, amelyet a saját alkalmazásomban is felhasználok. Az előbbi két rendszeren saját méréseket is végzek.

Fontos különbség, hogy míg a WaveNet classifier csupán zárthalmazú beszélőfelismerésre képes, a voicemap más elven, hangvektorokkal működik, ezért támogatja a nyílthalmazú beszélőfelismerést.

3.1. Modern beszélőfelismerés

A következő fejezetben bemutatom a 2017-2019-ben publikált korszerű beszélőfelismerő rendszereket és az általuk elért eredményeket. Mivel ez a kutatási terület jelenleg népszerű és folyamatosan újabb eredményeket érnek el, a bemutatott rendszereket a hivatkozások száma, a dátum és a *www.paperswithcode.com* oldal rangsorolása alapján választottam ki.

Az eredmények általános összehasonlítása több ok miatt is nehéz feladat. A cikkekben a legjobb eredmények feltüntetése miatt relatív eredményeket tartalmaznak egy-egy korábban publikált rendszerhez képest. Ezen kívül változik, hogy zárt vagy nyílt-halmazú beszélőfelismerésről, beszélő azonosításról vagy beszélő-ellenőrzésről van-e szó. Változik maga a teszthalmaz nehézsége is: etnikai, nemi, foglalkozásbeli diverzitás, a hangminták hossza, a hangfelvétel minősége, van-e irreleváns zaj, stb. Továbbá a legtöbb cikk nem biztosít letölthető modellt, konfigurációt, szkripteket az előfeldolgozáshoz, így azonos környezethez reprodukálni kellene őket.

3.1.1. Deep Speaker: an End-to-End Neural Speaker Embedding System

A Deep Speaker (2017) egy neurális hálózatokon alapuló embedding rendszer. Az *embedding* kifejezés alatt valamilyen leképzést értünk. Például amikor hangokból hangvektorokat készítünk. A rendszer a hangmintákat egy hipergömbre vetíti, ahol azoknak a hasonlóságát koszinusz távolsággal méri [13].

A leképzések előnye, hogy felhasználástól függetlenek, módszertől függően használhatók beszélőazonosításra, identifikációra és diarizációra is.

Jellemző kinyerésre ResNet és GRU (LSTM változat) architektúrákkal próbálkoztak. Tanításhoz *triplet loss* veszteséget használtak, és a hangvektorok között koszinusz távolságot számoltak.

A Deep Speaker által elért eredményeket a korábbi legkorszerűbb *i-vektoros* rendszerekhez mérték. Ezeknek az általános felépítése a következőképpen néz ki:

- Statisztikák gyűjtése
- i-vektorok leképzése (jellemző kinyerés)
- osztályozás (PLDA)

Ezek a rendszerek GMM-UBM modellt használtak a statisztikák kinyerésére, amiket jellemzőkkel (pl. MFCC) optimalizáltak. A GMM-UBM helyett később DNN alapú megoldással is próbálkoztak. A GMM-UBM kimeneteként kapott sokdimenziós statisztikákat ezután alacsony dimeziójú i-vektorokká alakítják, amit aztán PLDA modellel osztályoznak és verifikációra használják. Jelenleg a mai korszerű rendszerek az első két lépést összevonják és CNN-eket használnak alacsony dimenziós jellemzőkinyerésre.

Rendszer	**EER (%)**	**ACC (%)**
DNN i-vector baseline	13,79	51,72
ResCNN, softmax	6,13	81,95
ResCNN, triplet	2,69	86,21
ResCNN, softmax (pre-train) + triplet	2,23	90,53
GRU, softmax	5,42	83,05
GRU, triplet	3,23	84,19
GRU, softmax (pre-train) + triplet	2,77	89,50

3.1. táblázat. Mandarin nyelvű szöveg-független azonosítás eredményei tanításra és tesztelésre a Train50k és Eva200 adathalmazokat használva.

A legjobb architektúrának a *ResCNN* bizonyult softmax előtanítással és triplet lossal tanítva. Ezzel a 3.1 szerint Eva200 adathalmazon *90,53%*-os eredményt értek el. A Train50k, Train250k és Eva200k az UID adathalmaz részei.

Rendszer	**EER (%)**	**ACC (%)**
ResCNN on Train50k	2,23	90,53
ResCNN on Train250k	1,83	92,58
GRU on Train50k	2,77	89,50
GRU on Train250k	2,35	90,77

3.2. táblázat. Szöveg-független azonosítás eredményei.

A 3.2 szerint pedig nagyobb adathalmazzal való tanítás nagyobb pontosságot és alacsonyabb EER-t eredményezett. A legjobb eredményt a RestCNN-el érték el Train250k-val tanítva.

Az eredmények a korábbi i-vektoros rendszerekhez képest a Deep Speaker 50% csökkentette az EER-t, és pontosságban 60%-os javulást ért el.

3.1.2. Speaker Recognition from Raw Waveform with SincNet

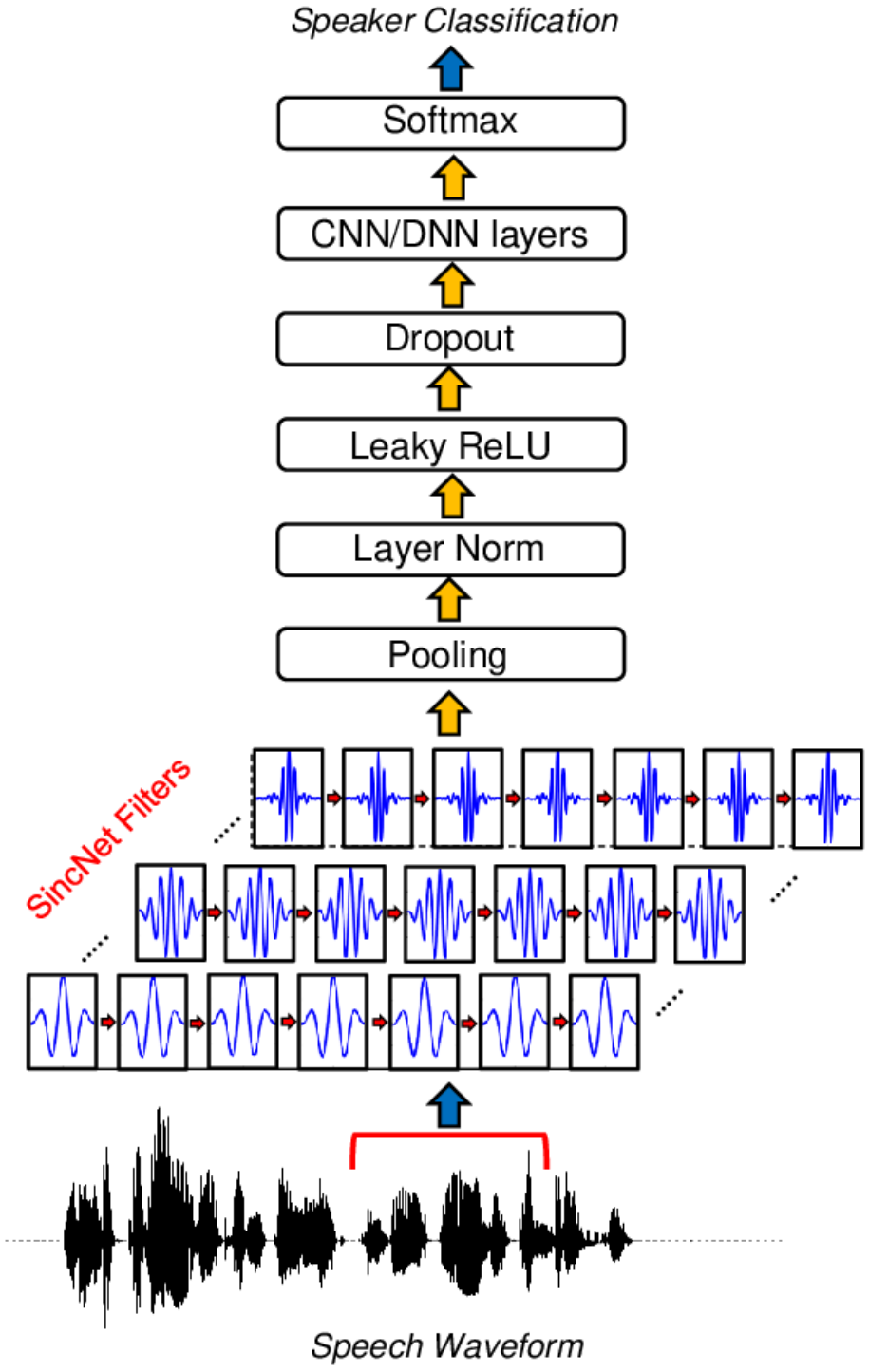
A SincNet (2018) is a korábban használt i-vektoros rendszerek helyett mutat be egy új CNN hálózatot. A CNN-ek előnye, hogy a korábbi emberi hallás modellje alapján alkotott jellemzők (pl. MFCC) helyett a hálózat maga tanulja meg a hang alacsony szintű reprezentációját nyers hullámformákból. Mivel a cikk a CNN-t kiegészíti hangszűrőkkel és a jelfeldolgozásban használatos fogalmakat használ, először ismertetem ezeket [14]:

- Beszéd frekvencia: Egy periodikus hullámforma esetében a frekvencia a periódusidő reciproka, azza megmondja, hogy egységnyi idő alatt hány periódus ment végbe. Az emberi hang esetében a komplex hullámforma Fourier analízissel felbontható trigonometrikus (periodikus) függvények összegére. A Fourier transzformáció segítségével az idő-amplitúdó doménből frekvencia-amplitúdót képzünk, így a frekvencia komplex jel esetén is értelmet nyer.
- Sávszélesség: Hang esetében általánosan a legalacsonyabb és legmagasabb frekvencia közötti különbség.
- Cutoff frekvencia: A hangszűrők legfontosabb paramétere. Például aluláteresztő szűrő esetében a cutoff fölötti frekvenciákat a szűrő tompítja, nem engedi át.
- Band pass filter: Olyan szűrő, ami csak a cutoff frekvencia környezetében lévő frekvenciákat engedi át (egy alul -és felüláteresztő szűrő kombinációja).

A SincNet egy új CNN architektúra, amelyben az első réteg ún. paraméterezhető sinc függvényekkel van kiegészítve, amelyek band pass szűrőket implementálnak.

A korábban használt kézzel fabrikált jellemzők, mint az MFCC vagy FBANK esetében semmi sem garantálja, hogy optimálisak minden beszélőfelismerő feladatra. A cikk szerint lehetséges, hogy ezek a jellemzők nem veszik figyelembe a hangmagasságot és a

formánsokat. A feltevésük szerint hullámforma feldolgozásánál a legfontosabb a CNN első rétege. Ezért a SincNet a hullámformákat sinc függvényekkel konvolválja, amelyek paraméterei az alacsony és magas cutoff frekvencia.



3.1. ábra. SincNet architektúra [14].

A SincNetet a TIMIT (462 beszélő) és Librispeech (2484 beszélő) adatbázisokkal tanították. A beszédminták elejéről a szüneteket eltávolították, és LibriSpeech esetén azokat a mintákat, amelyek tartalmaztak legalább 125 ms hosszú szünetet, annak mentén

feldarabolták. A TIMIT adatbázisnál egy beszélőtől 5 mintával tanítottak és 3 mintával teszteltek, míg LibriSpeechnél véletlenszerűen választott 12-16 másodperces mintákkal tanítottak és 2-6 másodpercesekkel teszteltek.

Rendszer	**TIMIT**	**LibriSpeech**
DNN-MFCC	0,99	2,02
CNN-FBANK	0,86	1,55
CNN-Raw	1,65	1,00
SINCNET	0,85	0,96

3.3. táblázat. Szöveg-független azonosítás eredményei adatbázisok szerint.

A 3.3-es táblázat a CER (Classification Error Rate [%]) hiba rátát mutatja. E szerint a SincNet jobban teljesít az MFCC, FBANK és nyers hullámformákkal tanított CNN-eknél mindkét adatbázis esetében, bár látható, hogy a sima CNN és a SincNet között minimális a különbség.

Rendszer	**d-vector**	**DNN-class**
DNN-MFCC	0,88	0,72
CNN-FBANK	0,60	0,37
CNN-Raw	0,58	0,36
SINCNET	0,51	0,32

3.4. táblázat. Szöveg-független ellenőrzés (verifikáció) eredményei.

A SincNettel nem jellemző vektorokat képeztek és a köztük lévő távolságot használták a hasonlóság mérésére, hanem osztályozással használják. Ebben az esetben zárt-halmazú beszélőazonosításról beszélünk. Ez sokkal kevésbé flexibilis, mert új beszélő esetén mindig újra kell tanítani a hálózatot, de cserébe jobb teljesítményre képes. A 3.5 ábrán egy d-vector nevű hangvektorokat képző neurális hálózatot hasonlítottak össze a SincNettel. Beszélőellenőrzés esetén a SincNet sokkal jobb EER-t ért el.

3.1.3. VoxCeleb2: Deep Speaker Recognition

A VoxCeleb1 és VoxCeleb2 (2018) nagy méretű, beszélőfelismerés céljából készült adatbázisok. Az előbbi több mint 1000, utóbbi pedig több mint 6000 hangmintát tartalmaz hírességektől. Az adatbázisok felépítésének részleteit a 2.1.4 és 2.1.5 fejezetek ismertetik. Mindkét adatbázissal végeztek méréseket különböző neurális hálózatokkal, de a legjobb eredményeket illetve nyílt-halmazú beszélőfelismerést a VoxCeleb2-vel érték el [12].

Bemutatják a VGGVox nevű CNN alapú neurális hálózatot, amelyet spektrogramokkal tanítottak és hangvektorokat hoz létre. Előnye ennek a megoldásnak, hogy nyílt-halmazú beszélőfelismerésre használható, új beszélő esetén nem szükséges újra tanítani a hálózatot.

Módosított VGG-M és ResNet architektúrákkal kísérleteztek. A modelleket contrastive loss veszteségfüggvénnyel tanították és előtanításra egy softmax réteget használtak keresztentrópiával, ami javította a modell teljesítményét. Tanító adathalmaznak a VoxCeleb2 adatbázist használták és a tesztelést a VoxCeleb1-el végezték. A méréseket EER és a 3.1 detektálási költség függvény szerint értékelték ki,

$$C_{det} = C_{miss} \cdot P_{miss} \cdot P_{target} + C_{fa} \cdot P_{fa} \cdot (1 - P_{target}) \tag{3.1}$$

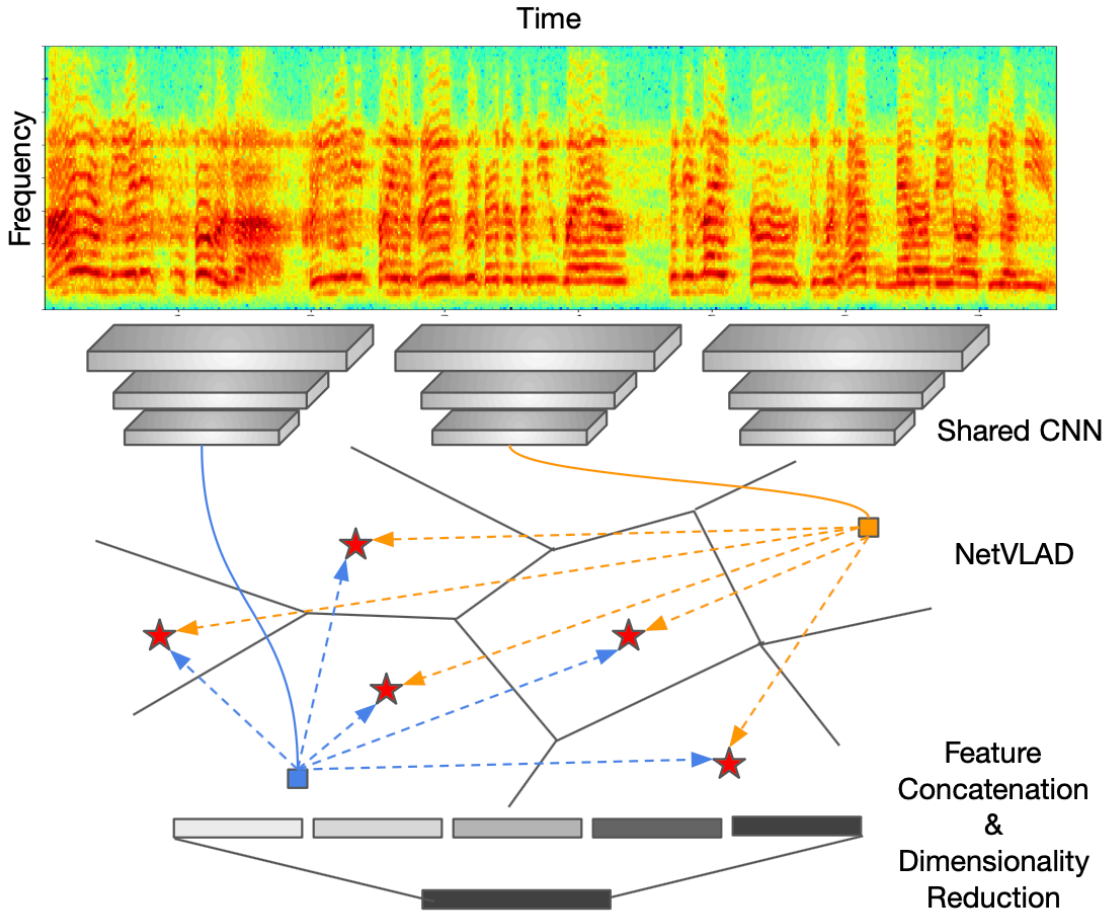
ahol *C_miss* és *C_fa* alkalmazás specifikus paraméterek a hamis negatív és hamis pozitív hibák költsége, *P_miss* és *P_fa* pedig a hibák előfordulásának valószínűségét jelzik. *P_target* annak a valószínűsége, hogy a rendszer két olyan mintát hasonlít össze, amely egy embertől származik.

Rendszer	**Tanítóhalmaz**	C_{min}^{det}	**EER**
I-vectors + PLDA	VoxCeleb1	0,73	8,8
VGG-M (Softmax)	VoxCeleb1	0,75	10,2
VGG-M	VoxCeleb1	0,71	7,8
VGG-M	VoxCeleb2	0,609	5,94
ResNet-34	VoxCeleb2	0,549	4,83
ResNet-50	VoxCeleb2	0,429	3,95

3.5. táblázat. Beszélő ellenőrzés (verifikáció) eredményei.

3.1.4. Utterance-level Aggregation for Speaker Recognition in the Wild

A 2019-es cikk két fő témája egy olyan neurális hálózat létrehozása, amely képes különböző hosszúságú hangmintákból jellemző vektorokat képezni és olyan hangminták vizsgálata amelyek irreleváns zajokat is tartalmaznak [15].



3.2. ábra. Neurális hálózat architektúra [15].

A hálózat három részből áll. Először egy módosított ResNet architektúrát használnak a keretszintű vektorok előállítására spektrogramokból. Az ábrán látható megosztott CNN-t hívjuk trönk hálózatnak. A keretszintű vektorok aggregálását a NetVLAD illetve GhostVLAD (Vector of Locally Aggregated Descriptors módszeren alapuló) hálózatokkal végzik. Legvégül egy FC (Fully Connected) réteget használnak a dimenzió 512-re csökkentésére.

A teljes modell end-to-end tanítására a VoxCeleb2 adatbázist használták és tesztelésként beszélőellenőrzést (verifikációt) mértek a VoxCeleb1-en.
A 3.3 táblázat mutatja a verifikációval mért EER-eket az akkori legkorszerűbb rendszerekkel szemben. Az addigiaknál lényegesen jobb EER-t értek el ezzel a módszerrel.

	Front-end model	Loss	Dims	Aggregation	Training set	EER (%)
VoxCeleb1 test set						
Nagrani *et al.* [2]	I-vectors + PLDA	–	–	–	VoxCeleb1	8.8
Nagrani *et al.* [2]	VGG-M	Softmax	1024	TAP	VoxCeleb1	10.2
Cai *et al.* [15]	ResNet-34	A-Softmax + PLDA	128	TAP	VoxCeleb1	4.46
Cai *et al.* [15]	ResNet-34	A-Softmax + PLDA	128	SAP	VoxCeleb1	4.40
Cai *et al.* [15]	ResNet-34	A-Softmax + PLDA	128	LDE	VoxCeleb1	4.48
Okabe *et al.* [13]	TDNN (x-vector)	Softmax	1500	TAP	VoxCeleb1	4.70
Okabe *et al.* [13]	TDNN (x-vector)	Softmax	1500	SAP	VoxCeleb1	4.19
Okabe *et al.* [13]	TDNN (x-vector)	Softmax	1500	ASP	VoxCeleb1	3.85
Hajibabaei *et al.* [19]	ResNet-20	A-Softmax	128	TAP	VoxCeleb1	4.40
Hajibabaei *et al.* [19]	ResNet-20	AM-Softmax	128	TAP	VoxCeleb1	4.30
Chung *et al.* [3]	ResNet-34	Softmax + Contrastive	512	TAP	VoxCeleb2	5.04
Chung *et al.* [3]	ResNet-50	Softmax + Contrastive	512	TAP	VoxCeleb2	4.19
Ours	Thin ResNet-34	Softmax	512	TAP	VoxCeleb2	10.48
Ours	Thin ResNet-34	Softmax	512	NetVLAD	VoxCeleb2	3.57
Ours	Thin ResNet-34	AM-Softmax	512	NetVLAD	VoxCeleb2	3.32
Ours	Thin ResNet-34	Softmax	512	GhostVLAD	VoxCeleb2	**3.22**
Ours	Thin ResNet-34	AM-Softmax	512	GhostVLAD	VoxCeleb2	3.23
Ours (cleaned †)	Thin ResNet-34	Softmax	512	GhostVLAD	VoxCeleb2	3.24
VoxCeleb1-E						
Chung *et al.* [3]	ResNet-50	Softmax + Contrastive	512	TAP	VoxCeleb2	4.42
Ours	Thin ResNet-34	Softmax	512	GhostVLAD	VoxCeleb2	**3.24**
Ours (cleaned †)	Thin ResNet-34	Softmax	512	GhostVLAD	VoxCeleb2	**3.13**
VoxCeleb1-H						
Chung *et al.* [3]	ResNet-50	Softmax + Contrastive	512	TAP	VoxCeleb2	7.33
Ours	Thin ResNet-34	Softmax	512	GhostVLAD	VoxCeleb2	**5.17**
Ours (cleaned †)	Thin ResNet-34	Softmax	512	GhostVLAD	VoxCeleb2	**5.06**

3.3. ábra. Neurális hálózat architektúra [15].

3.2. A hangminták hossza

Az eddigi rendszerekre hagyatkozva elmondható, hogy körülbelül 1-15 másodperces mintákkal tanították a neurális hálózatokat. Ez függ a beszédadatbázisban lévő hangminták méretétől, az előfeldolgozástól, és az határozza meg, hogy hiperparaméter optimizáció során mi bizonyul a legjobb értéknek. Ez rendszerenként változó. A 3.1 fejezetben ismertetett rendszerek és a voicemap kísérletek (3.5.5 fejezet) a következő eredményeket mutatták:

- Deep speaker: 1-5 másodperces hangmintákból képzett 20-dimenziós MFCC-t használtak tanításhoz.
- LibriSpeech: 12-16 másodperces mintákkal tanítottak
- VoxCeleb2: 3 másodperces mintákból képzett spektrogrammokat használtak.
- Speaker Recognition in the Wild: Átlagosan 7-8 másodperces mintákat használnak (VoxCeleb2 adatbázis)
- voicemap: 3 másodperces minta volt a legjobb optimizálás után.

3.3. Mérési környezet

A mérésekhez a Google Colaboratoryt használtam, amely gépi tanulási kutatások és oktatás céljából felhő alapú környezetet biztosít. Hozzáférni Google fiókkal lehet, egy felhasználó egy virtuális gépet és egy Jupyter Notebookot kap. Választhatunk, hogy a notebook a Python 2-es vagy 3-as verzióját támogassa, illetve, hogy CPU-n, GPU-n vagy TPU-n szeretnénk tanítani. A Tensor Processing Unit (TPU) a Google által fejlesztett gyorsító hardver egység Tensorflows tanításokhoz.

Az erőforrásokhoz fájlrendszert is kapunk. Ehhez hozzácsatolhatjuk a Google Drive fiókunkat, így elérhetjük a rajta tárolt adatokat. A notebookon keresztül elérjük a mögötte futó Linuxot is, lehet terminál parancsokat futtatni ha *!* vagy *%* jelöléssel kezdjük a sort. Ez lehetővé teszi azt is, hogy pl. kisebb méretű dolgokat *wget*tel töltsünk le vagy navigáljunk a fájlrendszerben, létrehozzunk új mappákat vagy töröljünk, áthelyezzünk fájlokat. Ugyan egyszerre több notebookot használhatunk, a fájlrendszer és a hardveres erőforrások ezek között megosztottak.

3.3.1. Adatok előfeldolgozása

Az előfeldolgozó szkript a TIMIT adatbázis esetében a hangmintákról eltávolítja a NIST Sphere fejlécet és a mondat előtti és utáni szüneteket. Ezután normalizálja a hangmintákat. A CMU Arctic esetében a hangminták alapból normalizálva vannak, ezért csak azonos méretűre kell vágni őket az egységes bemeneti dimenziók érdekében (ahogy a TIMIT esetében is).

3.4. WaveNet classifier

A *WaveNet classifier* egy módosított WaveNet architektúra beszélőidentifikációhoz.

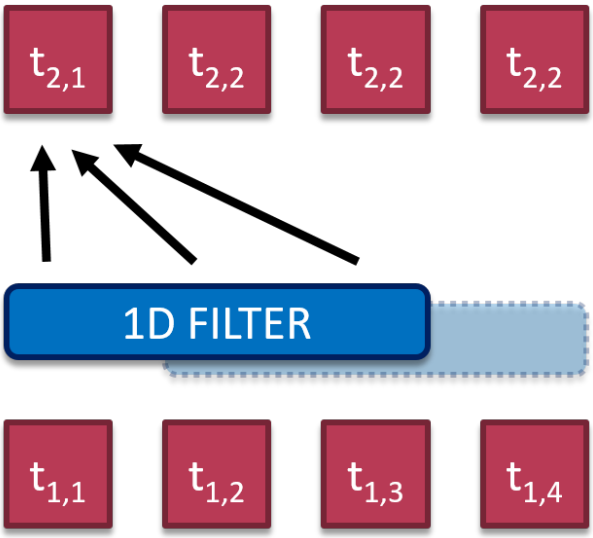
3.4.1. WaveNet

A WaveNet egy mély neurális hálózat audio hullámformák generálásához, amelyet a Google DeepMind publikált 2016-ban. Az ötletet az akkori felfedezések adták neurális autoregresszív generatív modellezés terén, amelyeket komplex eloszlások, például képek modellezésére használtak. Ezt felhasználva audio hullámformák generálásában új eredményeket értek el [16].

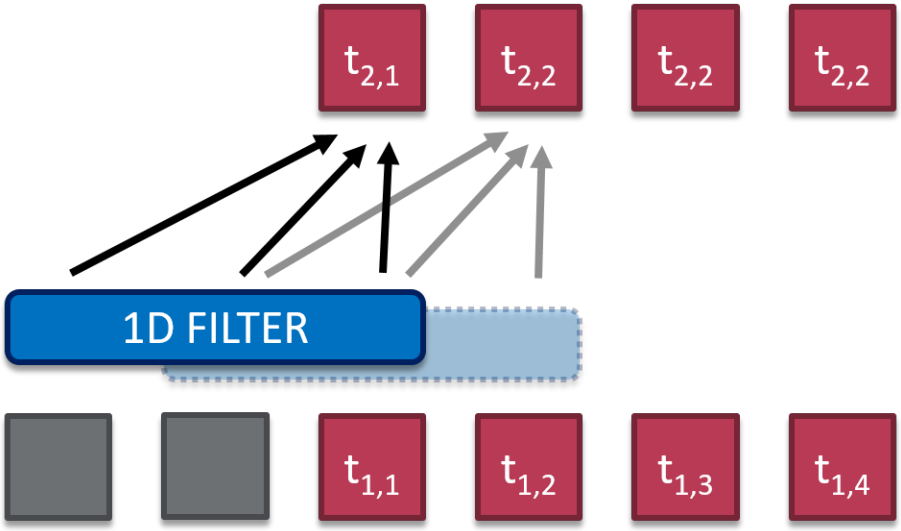
- Képes olyan természetes hangzású beszéd hullámformák generálására, amit korábban parametrikus vagy konkatenatív beszédszintézissel sosem értek el.
- Az audio hullámformák generáláshoz szükséges nagy receptív mezőt hatékonyan, nyújtott kauzális konvolúciókkal implementálja.
- Ha a modellt a beszélők identitásával tanítják, képes új hangok generálására.
- A zene generálás és a beszédfelismerés terén is ígéretesnek bizonyult.

3.4.1.1. Nyújtott kauzális konvolúció

A modell autoregresszív, vagyis a kimenete korábbi időpillanatokban felvett értékeitől függ. Ez azért fontos, mert a WaveNet egy generatív modell. A generált hullámforma t-edik időpillanatbeli értéke nem függhet jövőbeli értékektől. A kauzalitás legegyszerűbb implementációja ha legalább *kernel-1* méretű paddinget adunk a konvolúcióhoz.



3.4. ábra. 1D nem kauzális konvolúció.

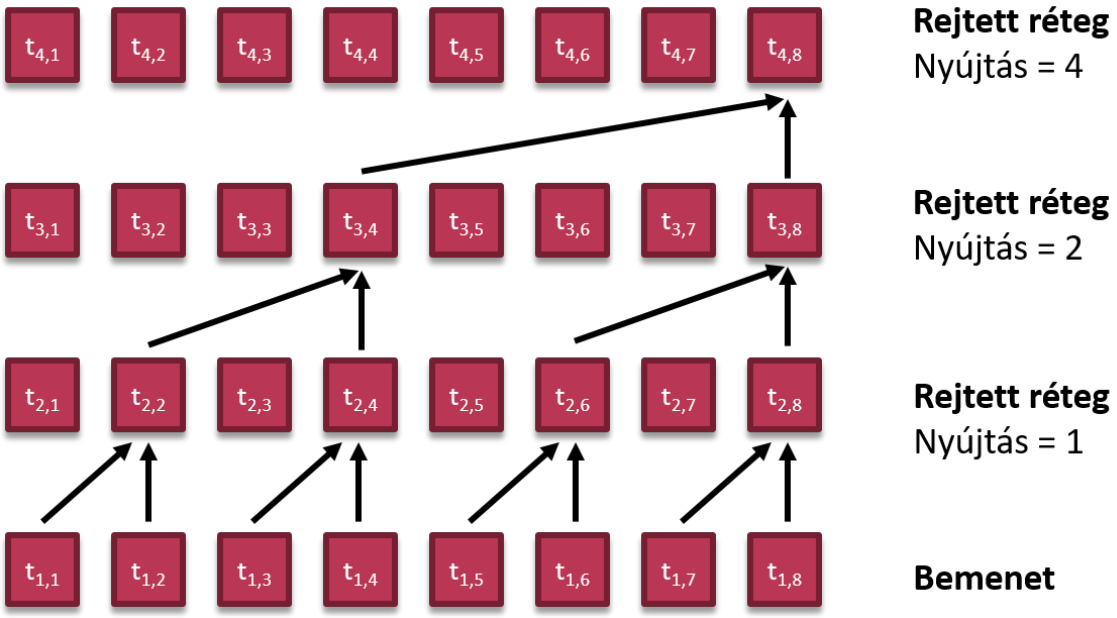


3.5. ábra. 1D kauzális konvolúció paddinggel.

A 3.4 ábra egy nem kauzális 1D konvolúciót mutat. A $t_{i,j}$ az i-edik rétegbeli j-edik neuron. Látható, hogy a $t_{2,1}$ jövőbeli időpillanatokból kap értékeket a szűrőn keresztül. Ennek megoldása a szűrő eltolása padding segítségével. Ezt szemlélteti a 3.5 ábra, ahol az egyes neuronok csak korábbi időpillanatokból kapnak értékeket.

A beszéd generálásánál a t-edik időpillanatban a hullámforma értéke a korábbi adatoktól függ. Ahhoz, hogy magas frekvenciájú, pl. 16 kHz frekvencián mintavételezett hangadattal tanítsuk a hálózatot nagy receptív mezőre van szükség. A receptív mező az a szélesség, amit a szűrő lát a bemenetből. 16 kHz esetén egy másodpercnyi jelet 16000 szám reprezentál. Ahhoz, hogy a hálózat helyesen jósolja meg a következő generált értéket, a hosszú távú dependenciákat figyelembe kell vennie, tehát a receptív mező méretét elég nagyra kell megválasztani. A probléma ekkora mezők esetén, hogy sok konvolúciós réteget igényelnek (egy korábbi időpillanatbeli adat plusz egy konvolúciós réteget igényel), ami növeli a számítási komplexitást.

A nyújtott konvolúciók erre adnak hatékony megoldást. A filter meghatározott távolságokkal kihagy valamennyi inputot, majd figyelembe vesz egyet. Egymás utáni rétegekben a nyújtási tényezőt exponenciálisan növelve a receptív mező is exponenciálisan fog nőni.



3.6. ábra. 1D nyújtott kauzális konvolúciós rétegek.

3.4.1.2. SoftMax eloszlás

A WaveNet SoftMax réteget használ a $p(x_t|x_1, \ldots, x_t)$ feltételes valószínűségi eloszlás modellezésére. Az audio jeleket általában 16 bites egészekkel kódolják, amelyek 65536 értéket vehetnek fel. Ebben az esetben a SoftMax rétegnek 65536 valószínűséget kell kimenetként adnia, melyek összege 1. A μ-law kvantálást alkalmazva a beszédjel 256 biten kódolható és később az inverz transzformációval jó minőségben visszaállítható.

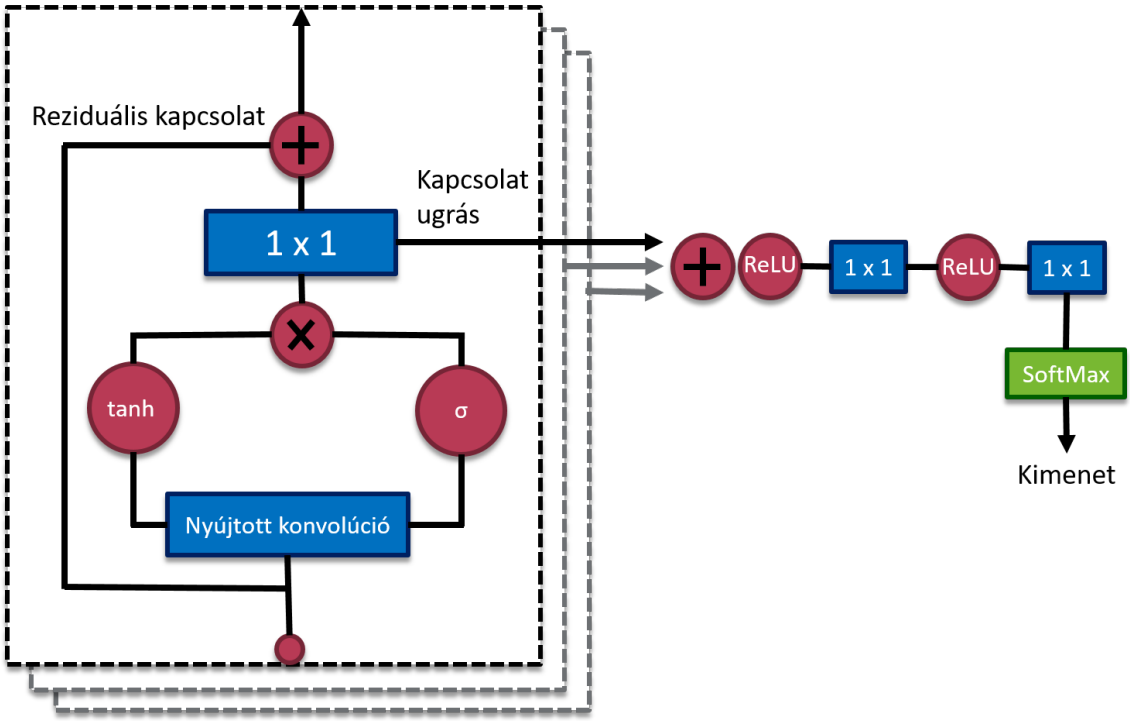
Az emberi hallás sokkal érzékenyebb alacsony amplitúdójú hangok kvantálási zajára, mint a magasabbakéra. Erre alapozva a *mu*-law kvantáló a jelet egy logaritmikus függvénnyel kvantálja úgy, hogy az alacsonyabb amplitúdójú jelek nagyobb felbontással (több bittel), a magasabbak pedig kisebbel lesznek kódolva. Ez növeli a SoftMax réteg hatékonyságát is, mert nagyobbak lesznek a valószínűségek közötti különbségek.

A tanítást a WaveNet klasszifikációs problémaként kezeli. A bemeneteket OneHot kódolással adjuk meg, a SoftMax réteg pedig az így kódolt egészekre ad valószínűségi eloszlást.

3.4.1.3. Reziduális blokkok

A WaveNet architektúra egymáshoz csatolt reziduális blokkokból és ún. kapcsolatugrásokból (skip-connection) épül föl. A reziduális hálózatok előnye, hogy orvosolják az elenyésző gradiens problémát, így sokkal mélyebb hálózat építhető.

Egy reziduális blokk bemenete egy 2x1-es konvolúciós rétegen megy keresztül. Balra egy *tanh*, jobbra egy szigmoid aktivációs függvényen haladnak át, majd elemenkénti szorzás és 1x1 konvolúció után egyrészt átugorja a reziduális kapcsolatot, illetve azzal együtt



3.7. ábra. WaveNet architektúra.

bemenetként szolgál a következő reziduális egységnek. Az 1x1 konvolúciós rétegek a dimenzionalitás változtatására szolgál. Külön 1x1 konvolúciós szűrők skálázzák a kimenetet a következő reziduális blokk bemenetére, és a kapcsolat-ugrásokhoz.

3.4.2. Módosított WaveNet architektúra

A módosított WaveNet architektúra segítségével a WaveNetet beszélőfelismerésre használhatjuk.

```
from WaveNetClassifier import WaveNetClassifier

wnc = WaveNetClassifier((96000,), (10,), kernel_size = 2, dilation_depth = 9,
                        n_filters = 40, task = 'classification')

wnc.fit(X_train, y_train, validation_data = (X_val, y_val), epochs = 100,
        batch_size = 32, optimizer='adam', save=True, save_dir='./')

y_pred = wnc.predict(X_test)
```

3.8. ábra. Wavenet classifier kód részlet.

A WaveNetClassifier objektum paraméterei:

- *input_shape*: Bemeneti dimenziók tuple formájában. Például ha a bemenet egy 6 s hosszú hullámforma 16 kHz-en mintavételezve, a bemeneti dimenziók (96000,)
- *output_shape*: Kimeneti dimenziók tuple formájában. Például ha 10 osztály szerint klasszifikálunk, a kimeneti dimenziókból képzett tuple (10,).

- *kernel_size*: A konvolúciós filter/kernel mérete a reziduális blokkokban.
- *dilation_depth*: A reziduális blokkok száma.
- *n_filters*: A konvolúciós filterek száma a reziduális blokkokban.
- *task*: Klasszifikáció vagy regresszió.
- *regression_range*: A regresszió céltartománya lista vagy tuple formátumban.
- *load*: Előző WaveNetClassifier betöltése (bool).
- *load_dir*: A betölteni kívánt modell könyvtára.

3.4.3. Eredmények

A WaveNet classifiert a 3.8 szerinti alapvető paraméterekkel, de 18 beszélővel a TIMIT és CMU Arctic beszédadatbázisokkal teszteltem. A TIMIT beszédkorpusszal csak kevés beszélő esetén ért el jó eredményt, több mint 20 beszélő esetén a modell nem tanult. Ennek valószínűsített oka, hogy a TIMIT adatbázis beszélőnként 10 hangmintát tartalmaz.

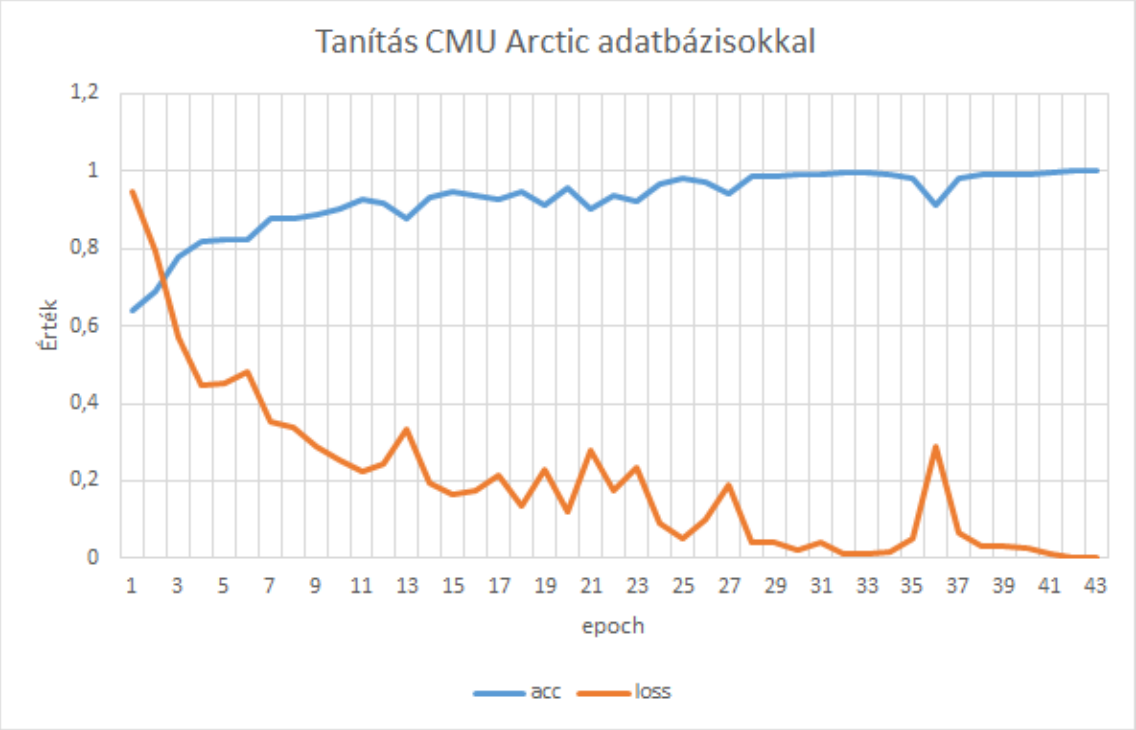
Beszélők száma	18
Minta/beszélő	100
Minta össz.	1800
Minta hossza	4000
Epochok száma	43
Hiba	0.0013
Pontosság	1.0

3.6. táblázat. Paraméterek CMU Arctic adatbázissal.

A TIMIT TRAIN 630 beszélőt tartalmaz hangmintákat. Először lefuttattam rajta egy módosított előfeldolgozó szkriptet (LibriSpeech respository része), ami levágja a szüneteket a hangminták elejéről és végéről a WRD fájlnak megfelelően, majd normalizálja az amplitúdókat. ezekből kiválogattam azokat, amelyek legalább 2,5 mp hosszúak voltak, és beszélőnként 2:1 arányban tanító és teszthalmazra daraboltam őket. Kevés, 10 beszélővel és 2,5 másodperces hangmintákkal tanítva, a teszt adathalmazon a hálózat 96.799 %-os pontosságot ért el.

A CMU Arctic adatbázisokat letöltöttem a Google Colaboratory-s fájlrendszerre és egy mappa alá mozgattam őket. Ehhez csak az adatbázis kódjára volt szükség, mindegyik azonos kezdetű url-en található. A CMU Arctic 16 bit-es wav fájlokat tartalmaz. Normalizálásként minden mintavételezett értéket leosztottam a maximális hosszal ($2^{16} - 1$), hogy -1 és 1 közötti értéket vegyenek fel.

Tanítómintáknak 2-3 másodperces részeket választottam. A megfelelő hosszú wav fájlokból kivágtam a [0:40000] részt, így pontosan $40000/16000 = 2,5$ s-os szeleteket kaptam. Mind a 18 adatbázisból kiválasztottam 100 ilyen mintát. Ezeket véletlenszerűen összekevertem és ez az 1800 elemű halmazzal tanítottam. Az epochok száma 200 volt, de mivel a tanítás a 43-44. epochnál már 1-es pontosságot és közel 0 veszteséget mutatott, leállítottam a folyamatot.



3.9. ábra. WaveNet tanítása CMU Arctic adatbázisokkal.

A teszthalmazba a tanítómintákon kívüli minimum 40000 hosszú wav fájlok kerültek, összesen 10152 minta. Az elvárt kimeneteket one-hot kódoltam és kiértékeltem a modellt a tesztadatokon. A felismerés pontossága 97, 3 % volt 18 ember esetén.

3.5. Voicemap

A *voicemap* egy nyílt forráskódú deep learning projekt beszélőfelismerő feladatok elvégzéséhez. A GitHub repository tartalmazza a modell kódját, a tanítást, különböző kísérleteket és egy hiperparaméter-optimizált, előre tanított modellt. Az implementációhoz *Kerast* illetve újabb verziókban *Pytorchot* használ [17].

A projekt célja, hogy későbbiekben egy olyan általános, pip által telepíthető csomag legyen, ami könnyen felhasználható beszélőfelismeréssel kapcsolatos feladatok elvégzésére.

3.5.1. A projekt általános felépítése

Mivel a saját alkalmazásomban felhasználtam és átalakítottam a projekt egyes részeit, röviden bemutatom a felépítését.

- *models.py*: A konvolúciós enkóder létrehozását és a sziámi hálózat (4.2.1 fejezet) építését végzi el.
- *librispeech.py*: Keras szekvencia osztály Librispeech beszédadatbázisból származó hangminták tárolására, előfeldolgozására és generálására tanításhoz. Fő paraméterei a következők:
  - *subsets*: Mely LibriSpeech adathalmazokat tartalmazza.

  - *seconds*: Az ennél rövidebb időtartamú hangmintákat figyelmen kívül hagyja.
  - *stochastic*: Sztochasztikus módban a mintákból véletlenszerűen vágjuk ki a *seconds* hosszúságú darabot.
  - *pad*: Padding esetén a rövidebb hangmintákat nullával feltöltve a megadott méretre alakíthatjuk. Sztochasztikus mód esetén véletlenszerűen oszlik el a hangminta előtti és utáni nullák száma.

  A két legfontosabb függvénye:

  - *build_verification_batch*: A sziámi hálózat teszteléséhez generál olyan batchet, amelyben az ugyanattól és az eltérő beszélőktől származó hangmintapárok száma azonos (50%-50%). A batch egy eleme két hangmintából és egy címkéből áll, ami jelzi, hogy egyazon vagy más-más beszélőktől származnak.
  - *build_n_shot_task(k, n)*: Visszaad egy segédhalmazt n mintával minden egyes k beszélőhöz és visszaad egy mintát, amiről a modellnek el kell döntenie, hogy melyik beszélőhöz tartozik.

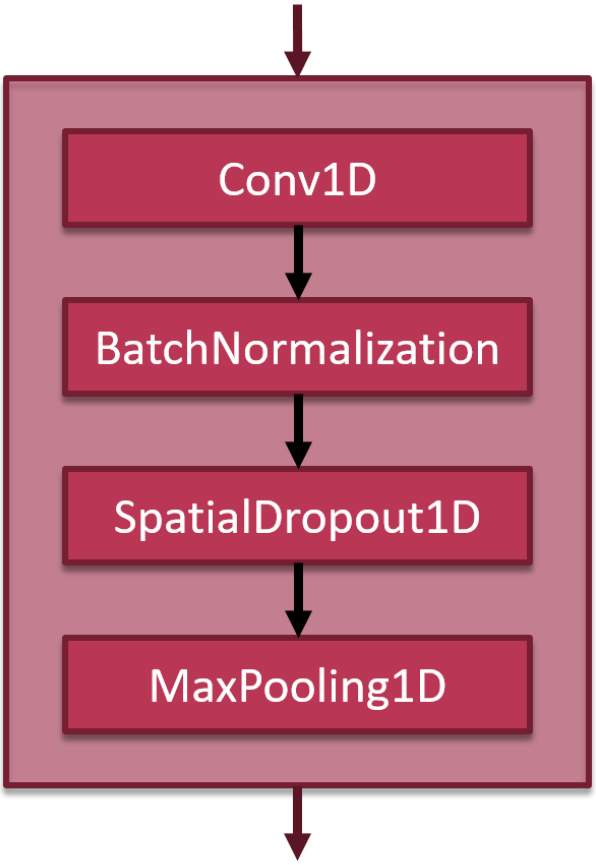
- *utils.py*: Fő feladata az adatok preprocesszálása és az *n_shot_task_evaluation* függvény által az *n-shot k-way* feladat kiértékelése (részletesen a 4.1 fejezet ismerteti). A few-shot kiértékeléshez alábbi argumentumokat adhatjuk meg:

  - *model*: A tesztelendő modell.
  - *dataset*: A tesztmintákat tartalmazó adathalmaz objektum.
  - *preprocessor*: Előfeldolgozó függvény a minták csökkentett mintavételezésére és standardizálására.
  - *num_task*: A feladatok száma.
  - n: Hány hangminta tartozik egy beszélőhöz a segédhalmazban.
  - k: Ennyi beszélő, azaz osztály van a segédhalmazban.
  - *network_type*: Sziámi vagy klasszifikációs.
  - *distance_metric*: Sziámi hálózat esetén a hangvektorok közötti távolságmetrika.

3.5.2. Az implementált modellek

A projekt két modellt vizsgál; egy sziámi neurális hálózatot (a konvolúciós sziámi hálózat részletes leírása a 4.2.1 fejezetben található) és egy sima konvolúciósat klasszifikációval. Mindkét hálózat alapja alapja egy konvolúciós enkóder, amely a nyers hangmintákból kinyeri a jellemzőket és hangvektorokat állít elő belőlük. Az enkóder hálózat a több egymást követő konvolúciós blokk áll. Egy konvolúciós blokk felépítése a 3.10 ábrán látható. Ezek egy konvolúciós rétegből és regularizációs rétegekből tevődnek össze.

- Conv1D: 32-es méretű filterekkel, ahol a filterek száma arányosan nő azzal, hogy a blokk hányadik a sorban. Az aktivációs függvény *ReLu*.
- BatchNormalization: Batchenként normalizálja az előző réteg kimeneteit úgy, hogy az átlag 0-hoz, a szórás 1-hez közelítsen. Regularizációra használják.
- SpatialDropout1D: Sima dropout réteg esetén az egyes neuronokat valamekkora valószínűséggel figyelmen kívül hagyjuk. Például egy [[1, 2, 3], [2, 3, 1]] tömbnek a kimenete sima dropout esetén lehet [[1, 0, 3], [0, 3, 1]], itt teljesen függetlenek egymástól a kinullázások. Spatial dropout esetén az adott dimenzió mentén mindent kinullázunk. Például [[1, 0, 3], [2, 0, 1]]. Regularizációra használják.



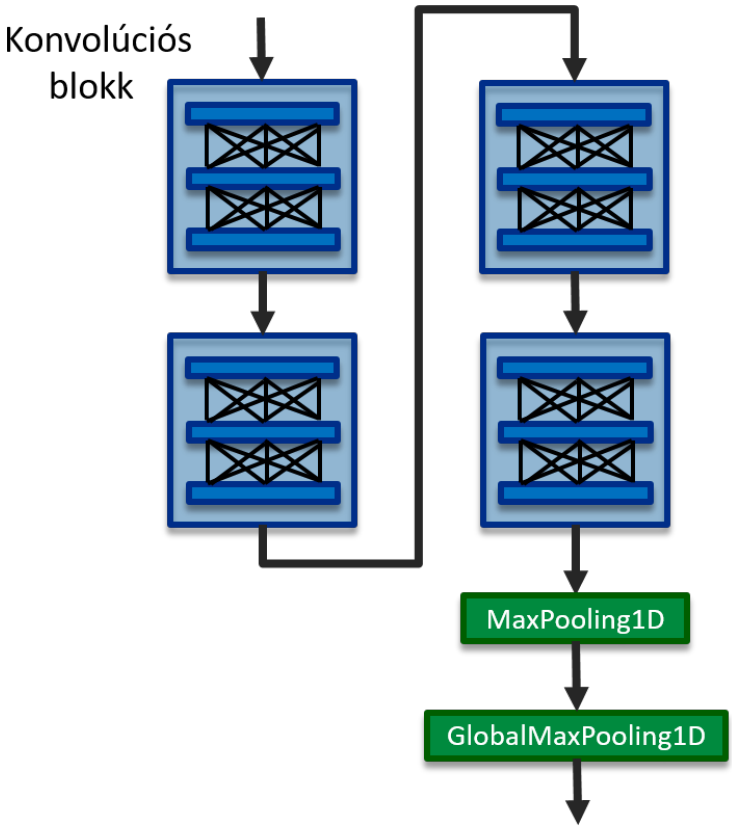
3.10. ábra. Egy konvolúciós enkóder blokk felépítése.

Négy ilyen konvolúciós blokk követi egymást, majd további regularizáció: egy *GlobalMaxPool1D* és egy *Dense* réteg a hangvektor méretével.

A sziámi hálózat alapja két enkóder hálózat, amelynek súlyai megosztottak, tehát úgy is tekinthetünk rá, hogy egy enkóder hálózatra két bemenetet adhatunk. A hangminták a hálózaton áthaladva jellemző vektorokká alakulnak. Ezt a részt a konvolúciós enkóder végzi. Ezután a két vektor közötti távolságmetrikát a sziámi hálózat kiszámolja. Ezen távolság alapján számolja ki a veszteséget és javítja a súlyokat.
Az implementált távolságmetrikák a következők:

- *weighted_l1*: Az eredeti one-shot cikk szerinti távolságmetrika. v_1 és v_2 vektorokra $\sqrt{v_1 - v_2}$.
- *uniform_euclidean*: Euklideszi távolság két vektor között.
- *cosine_distance*: Koszinusz távolság, azaz a két vektor által bezárt szög koszinuszát méri.

A kiszámolt távolság ezután minden esetben áthalad egy szigmoid aktivációs függvényű *Dense* rétegen, ami 0 és 1 közé nyomja az eredményt.



3.11. ábra. A konvolóciós enkóder felépítése.

3.5.3. Beszédadatbázisok és generátorok

A voicemap tanításhoz és teszteléshez a *LibriSpeech* adatbázis *dev-clean*, *train-clean-100* és *train-clean-360* adathalmazokat használja, amelyek sorban 40, 251 és 921 különböző beszélőtől tartalmaznak nyers hangfájlokat.

Egy adathalmazt egy adatgenerátor osztály reprezentál ami a keras.util.Sequence osztályból származik. Az adatgenerátorok nagy adathalmazok esetén hasznosak, amikor az egész adathalmaz nem fér bele a memóriába. Ilyen esetekben egyesével generálnak adatokat.

A *keras.util.Sequence* osztályból való leszármaztatás miatt az adatgenerátorban implementálni kell a *__len__* és *__get_item__* függvényeket. Előbbi az adathalmaz méretét adja vissza, utóbbi annak indexelését teszi lehetővé. Továbbá biztosítja, hogy egy epochon belül egy mintával csak egyszer tanítjuk a modellt.

3.5.4. Tanítás

A sziámi és a sima klasszifikációs hálózat nagyrészt közös paraméterekkel rendelkeznek. A fő különbség, hogy míg a sziámi veszteségfüggvénye bináris keresztentrópia, és a veszteség az alapján dől el, hogy a hangminta pár egyazon vagy más beszélőktől származik, a klasszifikációs hálózat kategórikus keresztentrópiát használ, tehát a hangmintát megpróbálja besorolni k osztály valamelyikébe k beszélő esetén. A közös paraméterek:

- hangminta hossza: 3 sec

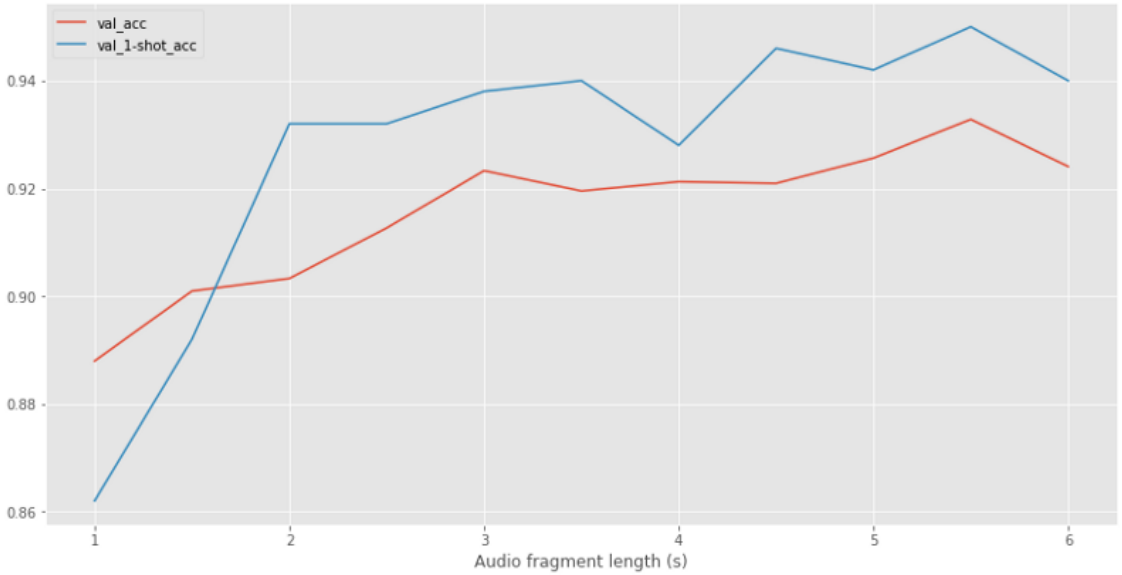
- batchsize: 64
- filterek száma: 128
- jellemző vektor dimenzió: 64
- dropout: 0
- steps_per_epoch: 500
- kiértékelési feladatok száma: 500
- n_shot_klasszifikáció: 1
- k_way_klasszifikáció: 5

Tehát mindkét modell optimizált hiperparamétereket használ, *Adam* optimizálót és veszteségfüggvényként keresztentrópiát. Egy epoch 500 batch iterációból áll és egy batch méret 64. Az epochok végén három callback fut le.

Minden epoch végén kiértékelés történik: 500 *1-shot 5-way* feladat átlagos eredménye jelzi a pontosságot. Amennyiben ez növekszik a modellről *checkpoint* készül (*ModelCheckpoint*). Továbbá ha a kiértékelés során a pontosság nem nő, a *ReduceLROnPlateau* csökkenti a tanulási rátát.

3.5.5. Kísérletek és optimalizálás

A repositoryban számos kísérlet található a legjobb teljesítmény elérésére. A *wide_vs_tall* szkript leméri, hogy a modell hogyan teljesít különböző hosszúságú hangmintákkal tanítva. A mérés alatt *1-shot 5-way* feladatokkal, azaz 5 különböző beszélőtől 1-1 beszédmintával validálta a modellt [18].



3.12. ábra. Voicemap: Regisztrációs fázis.

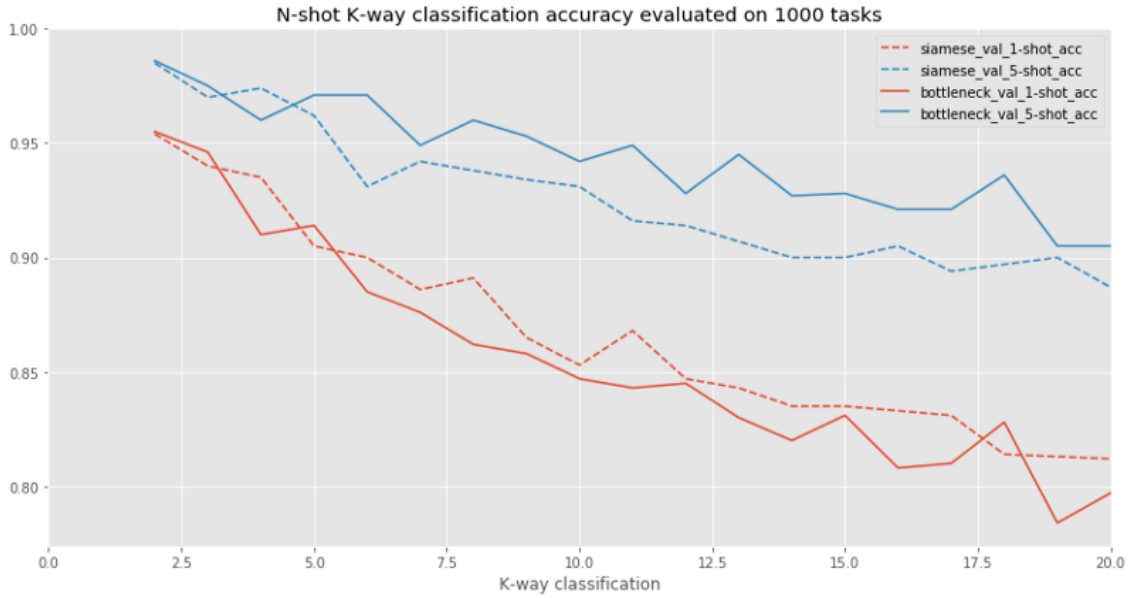
A 3.12 ábrán látható, hogy a hangminta méretét növelve a modell valóban jobban tanul, de a több adat miatt megnövekedik a tanítási idő és több memóriára van szükség. A grafikon mutatja, hogy 3 másodperc után a validáció stagnálni kezd, ezért az erőforrásokat

és a tanítási időt figyelembe véve ez tűnik a legjobb választásnak.

A *grid_search_siamese_network* szkript hiperparaméter optimizációt végez a sziámi hálózaton a filterek számát, a hangmintákból képzett vektorok hosszát és a dropoutot vizsgálva. A talált legjobb paraméterek:

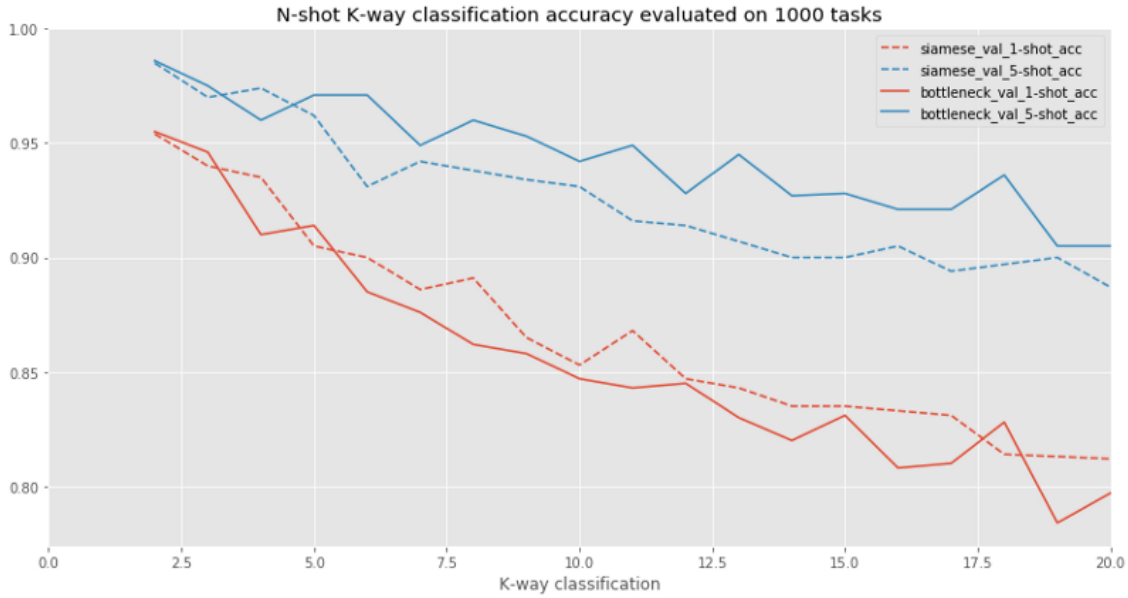
- filterek száma: 128
- jellemző vektor dimenzió: 64
- dropout: nincs

A *k_way_accuracy* kísérlet a hiperparaméter optimalizált modell teljesítményét méri le különböző *n-shot k-way* feladatokkal. A pontosságot a 3.14 ábra mutatja.



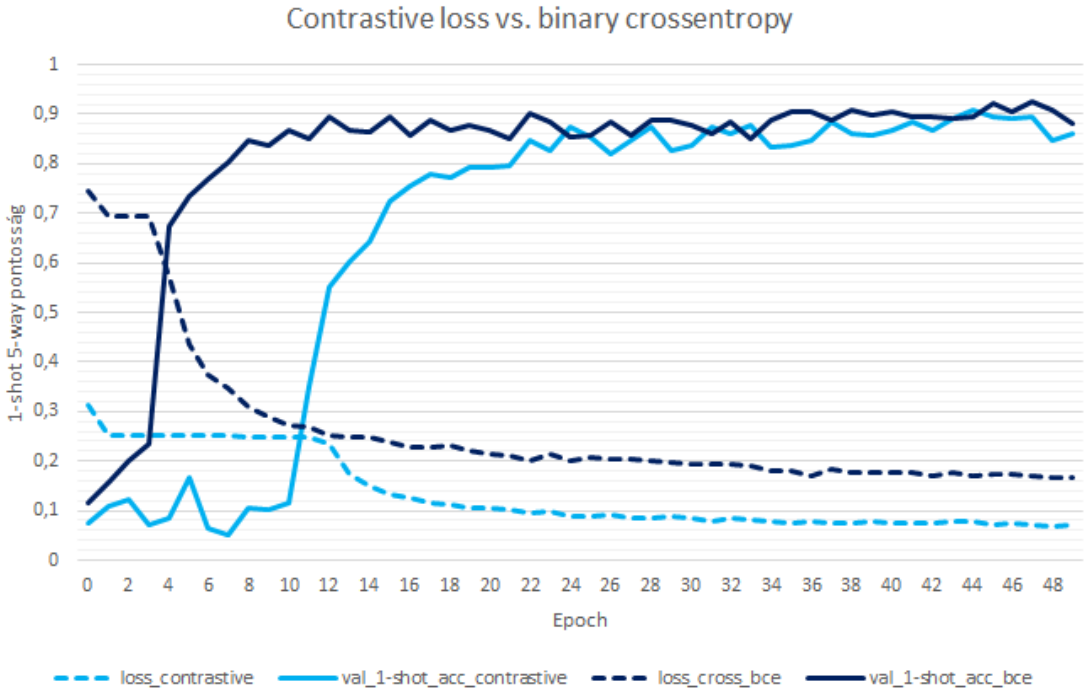
3.13. ábra. Voicemap: n-shot k-way pontosság.

1-shot k-way feladatok esetén a sziámi hálózat átlagosan jobban teljesít sima klasszifikációsnál, de *5-shot k-way* feladatok esetén már az utóbbi kerül fölénybe. Ez valószínűleg annak köszönhető, hogy 5 hangminta - beszélő párból a sima osztályozó hálózat már jobban meg tudta tanulni a beszélőket a sziáminál.



3.14. ábra. Voicemap: n-shot k-way pontosság.

Lemértem sziámi hálózat esetén a contrastive loss és bináris keresztentrópia veszteségfüggvények közötti különbséget. Mindkét modellt az optimizált hiperparaméterekkel 50 epochon keresztül tanítottam.



3.15. ábra. Voicemap: n-shot k-way pontosság.

A 3.15 grafikon mutatja, hogy a *1-shot 5-way* pontosság a 20. epochnál kezd 90% fele

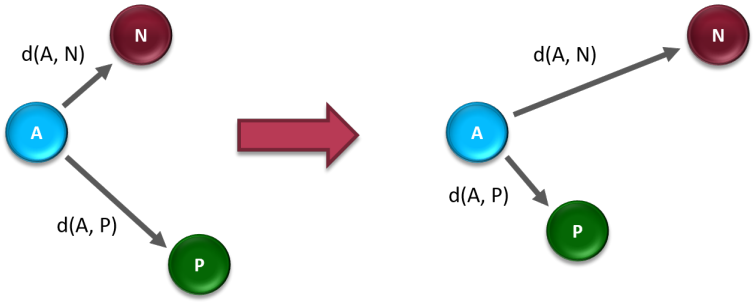
konvergálni. A maximális pontosság contrastive lossnál a 44. epochnál 90,8%, míg bináris keresztentrópia esetén a 47. epochnál 92,6%.

3.5.6. Saját kísérlet: Triplet loss

A sziámi hálózatoknak egy ismert költségfüggvénye a *triplet loss* függvény. Ezt a *gradient descent* módszerrel optimizálva tanítjuk a hálózatot. A triplet loss három bemenetet igényel, amelyek jelen példában a hangokból képzett jellemző vektorok. Az egyik egy rögzített hang vektora, ez az ún. *anchor*. A másik kettő pedig egy pozitív és egy negatív minta. A pozitív ugyanattól a beszéltőtől származik mint az *anchor* vektor, a másik különböző-től [19].

$$\mathcal{L}_{triplet}(A, P, N) = \max(d(A, P) - d(A, N) + m, 0) \tag{3.2}$$

A d a távolságfüggvény, ami lehet például euklideszi távolság. A *triplet loss* veszi az anchor és a pozitív meg az anchor és a negatív minta távolságainak különbségét, majd ezt eltolja az m küszöbértékkel. Ha az előbbi pozitív ezt veszi eredményül, egyébként nullát.



3.16. ábra. A triplet loss függvény csökkenti a távolságot a hasonló és növeli a különböző minták között.

Akkor jó a vektorok elhelyezkedése a metrikus térben, ha a hasonlók között a távolság kicsi, a különbözők között pedig nagy. Azt szeretnénk elérni, hogy $d(A, P) \leq d(A, N)$ fenn álljon. Átrendezve a $d(A, P) - d(A, N) \leq 0$ egyenletet kapjuk. Ezt kielégíti a $d(A, P) = 0$, $d(A, N) = 0$ megoldás. A másik triviális megoldás, ha a pozitív és negatív minta kódolása ugyanaz lenne, ekkor ugyanis $d(A, P) = d(A, N)$ így $d(A, P) - d(A, N) = 0$. Szeretnénk, ha a neurális hálózat nem nullvektorokkal vagy azonos vektorokkal kódolná az összes képet, ezért hozzáadunk egy m küszöbértéket az egyenlethez.

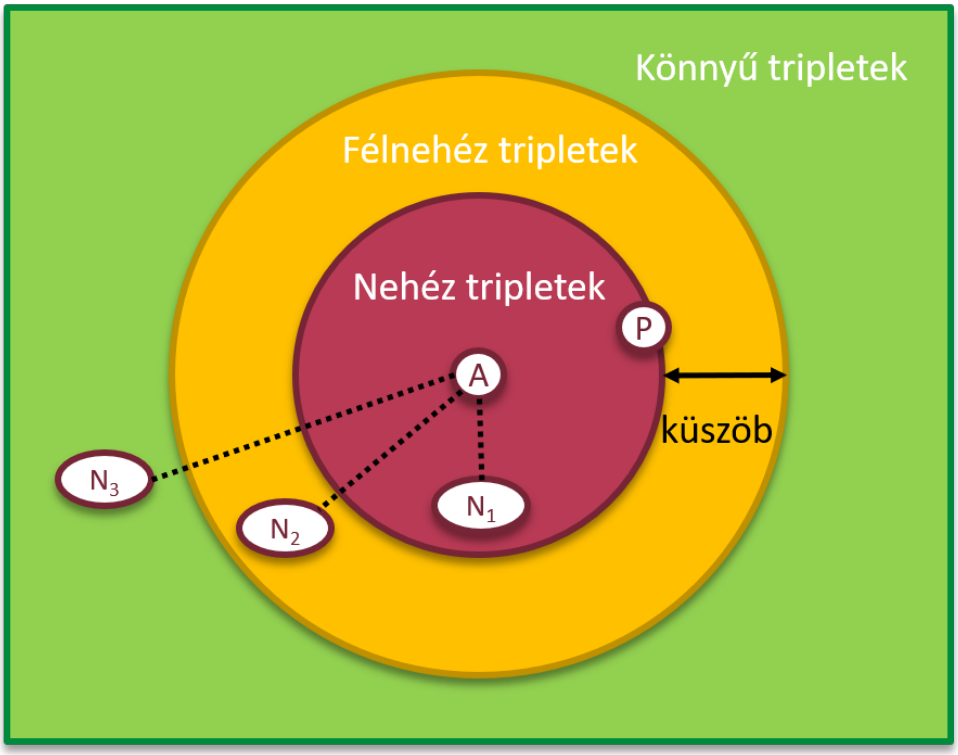
$$\begin{aligned} d(A, P) + m &\leq d(A, N) \\ d(A, P) - d(A, N) + m &\leq 0 \end{aligned} \tag{3.3}$$

Ideális esetben a $d(A, P) - d(A, N) + m$ negatív, ilyenkor a veszteség 0. Ha nem így van, a triplet loss ezt a veszteséget adja vissza. A költségfüggvény a tanítóhalmazban lévő tripletekre alkalmazott triplet lossok összege. Ezt minimalizálva a 3.16 ábrán látható távolságok csökkentése, növelése történik.

A tripletek kiválasztására többféle módszer létezik. A legegyszerűbb megoldás – amit én is használtam – a véletlenszerű kiválasztás. Emelett a tripleteket három kategóriába sorolhatjuk [19][20]. Ezt a 3.17 ábra szemlélteti:

- Könnyű (negatív) triplet: A 0 veszteségű tripletek, mivel $d(A, P) + m \leq d(A, N)$. (A 3.17 ábrán APN_3.)
- Nehéz (negatív) tripletek: A negatív közelebb van az anchorhoz mint a pozitív: $d(A, N) \leq d(A, P)$. (A 3.17 ábrán APN_1.)

- Félnehéz (negatív) tripletek: Olyan tripletek, ahol a negatív nincs közelebb az anchorhoz mint a pozitív, de mégis nagyobb a veszteség nullánál: $d(A, P) \leq d(A, N) \leq d(A, P) + m$. (A 3.17 ábrán APN_2.)



3.17. ábra. A tripletek három típusa.

Nehéz tripletek bányászása egy külön feladat és léteznek módszerek ún. offline (tanítás előtti) és online (tanítás közbeni) generálásra. Előnye, hogy gyorsabban konvergál a hálózat a veszteség minimuma felé. Ugyanakkor a FaceNet megjegyzi, hogy nehéz tripletekkel való tanítás az elején rossz lokális minimumhoz vezethet, ezért a félnehéz tripleteket ajánlja.

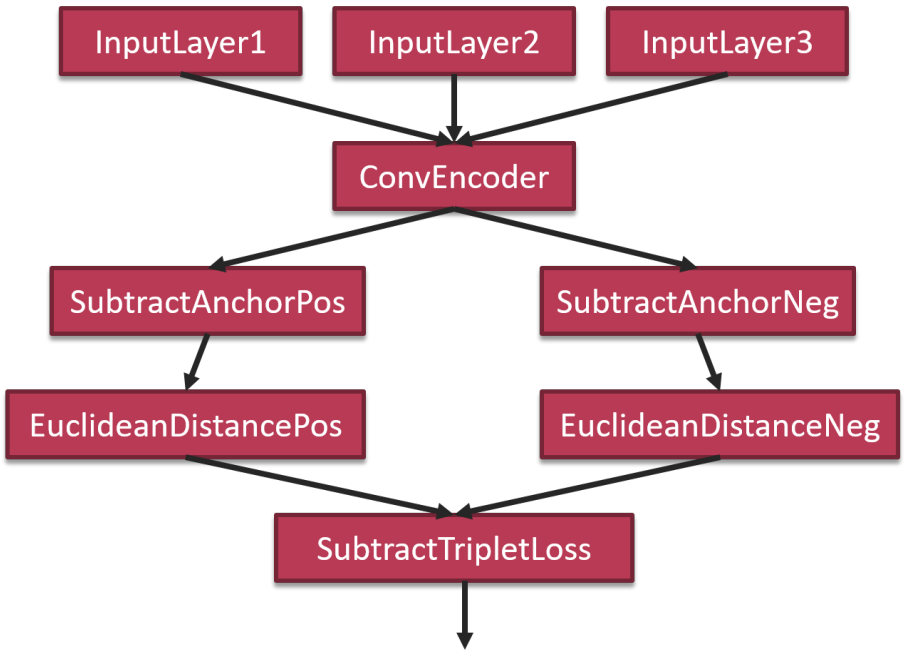
3.5.6.1. Voicemap implementáció

Létrehoztam egy bemenetként tripleteket fogadó neurális hálózatot *triplet loss* veszteségfüggvénnyel (a triplet loss veszteségfüggvény részletes leírása a 4.2.1 fejezetben található). Egy bemenet egy olyan hangminta hármas (triplet), ami két azonos és egy különbözőtől beszélőtől tartalmaz hangmintát.

A két azonos minta közül az egyik lesz az *anchor* érték. Ehhez mérten számoljuk ki a pozitív (azonos beszélőtől származó) és negatív (különböző beszélőtől származó) távolságokat.

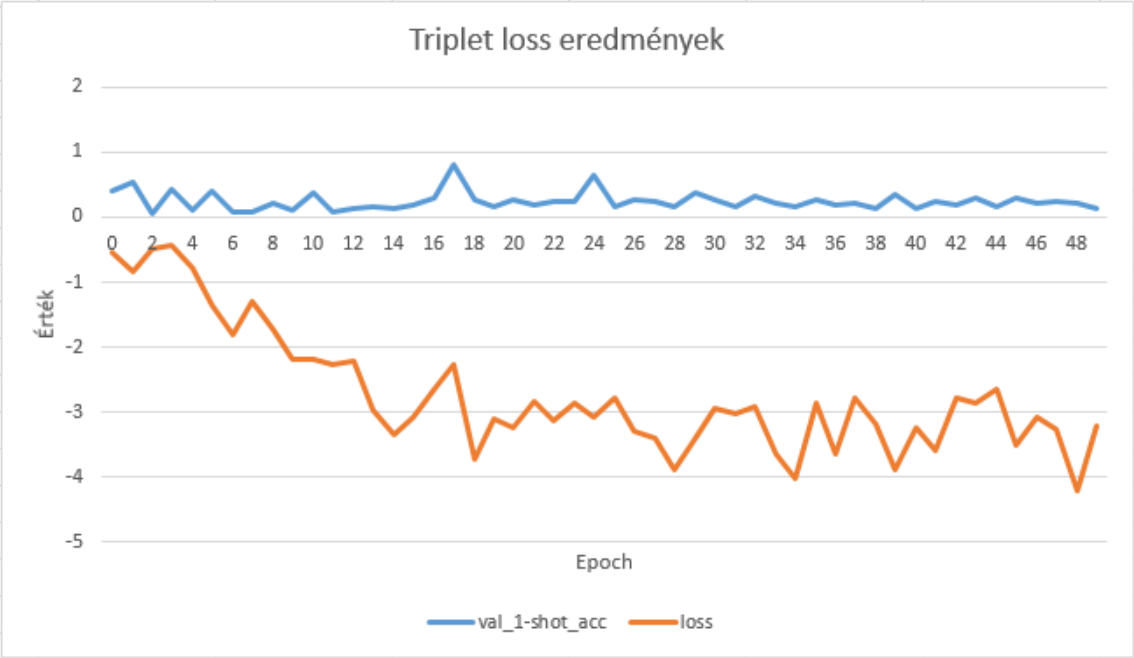
Az alapja a konvolúciós enkóder. Ehhez csatlakozik három bemenet, a kimenetére pedig két lambda réteg ami a pozitív és negatív távolságokat számolja ki, majd még két lambda réteg kiszámolja a vektorok hosszát. A szubtrakciós réteg ezután a két kiszámolt vektort kivonja egymásból, ahogy a 3.18 ábrán látható.

Tanításhoz a LibriSpeech adatbázist használtam. Ehhez módosítottam a *LibriSpeechDataset* osztályt úgy, hogy képes legyen tripletekből álló batchek generálására is.



3.18. ábra. Triplet hálózat.

Az *utils.py*-ben implementáltam magát a *triplet loss* veszteségfüggvényt, módosítottam a BatchPreProcessor wrapper osztályt, hogy egységesen kezelje a triplet alakú tanítóadatokat a sziámi és sima konvolúciós bemenetekkel együtt, majd kiegészítettem az *n_shot_task_evaluation* függvényt úgy, hogy képes legyen *k-way n-shot* feladat kiértékelést tripleteken is elvégezni.



3.19. ábra. Triplet lossal tanítás eredményei.

A tanítás során minden batch végén 1-shot 5-way kiértékelés történik, azaz 5 beszélőtől

egyelemű segédhalmazokkal próbálja a hálózat megtippelni, hogy melyik beszélőhöz tartozik a kérdéses hangminta.

A 3.19 grafikon azt mutatja, hogy az egyes epochok szerint hogyan változik a veszteség és a 1-shot 5-way pontosság. Látható, hogy a veszteség stagnálva, de csökken, ennek ellenére a validációs pontosság folyamatosan 0 és 1 között változik. A 17. és 24 epochnál megközelíti az 1-et, majd tovább romlik. A pontosság és a veszteség nem lassan, hanem egyáltalán nem konvergál, ezért feltételezhető, hogy nem a véletlenszerű tripletek kiválasztása a probléma. A tanítás nem volt sikeres.

4. fejezet

Meta learning

Ebben a fejezetben ismertetem a metatanítás fogalmát, a megközelítéseket. Megmutatom, hogy mit old meg a few-shot learning és ez hogyan felhasználható nyílt halmazú beszélőfelismerő rendszerekben.

> ”— Miért van szüksége a neurális hálózatoknak sok tanítómintára a jó teljesítéshez? Egy két éves gyerek miután látott pár autót képes felismerni azt. — Egy ekkora gyereknek volt két éve tapasztalatokat szerezni olyan dolgokról, amik nem autók voltak. Biztos vagyok benne, hogy ez fontos szerepet játszott a dologban.”
>
> (StackExchange [21] alapján)

Az emberek képesek hasznosítani a korábban szerzett tudást. Ha valaki megtanult biciklizni, utána sokkal könnyebben motorbiciklire ül. Felismerünk dolgokat úgy, hogy előtte csak néhányszor láttuk élőben vagy csak képen láttuk.

Az emberekkel szemben a neurális hálózatoknak két problémája van [22].

1. Nem tanulnak effektíven, sok tanítómintát igényelnek egy feladat megtanulásához. Például kézzel írt szöveg felismeréséhez karakterenként kb. 6000 minta szükséges.
2. Nem hasznosítják a korábban, más feladatok által megszerzett tudást.

Metatanítás alatt olyan tanító módszerek összességét értjük, amelyek felhasználják a korábban szerzett tudást és hasznosítják azt a későbbi feladatok során. A metatanuló modellek általánosítják a tapasztalataikat és könnyen adaptálódnak új feladatokhoz. A könnyű adaptáció alatt kevés tanítómintával végzett tanítást értünk, más néven finomhangolást.

4.1. Few-shot learning

A neurális hálózatok rengeteg tanítómintát igényelnek. A kevés tanítómintával tanított hálózatok túltanulnak; a tanítóhalmazon jó eredményt mutatnak, de csökken az általánosító képességük, így a teszthalmazon jelentősen rosszabb eredményt érnek el. Ugyanakkor ha sok tanítómintával tanítjuk a hálózatokat, az adott feladatokra már jól általánosítanak, de magukra a feladatokra tanítjuk túl őket. További hasonló feladatokra nem tudnak általánosítani [22].

A few-shot learning azzal a problémával foglalkozik, hogy hogyan építhetünk olyan modelleket, amelyek képesek új feladatokat gyorsan megtanulni. A modellt sokféle feladatra tanítjuk, de minden feladathoz kevés tanítóminta tartozik. A tanítás után új feladat esetén a modell kevés tanítómintával képes jól megtanulni azt.

A few-shot learningre való képességet az *n-way k-shot* feladattal szokták mérni [18].

1. A modell kap egy eddig nem látott osztályból egy tesztmintát.
2. Kap továbbá egy ún. segéd adathalmazt, ami az összes k eddig nem látott osztályból n mintát tartalmaz.
3. Az algoritmus el kell döntse, hogy melyik osztályba tartozik a tesztminta a segédhalmazból.

A beszélőfelismerő rendszereket tekintve nagyon fontos, hogy a neurális hálózat rugalmas legyen a tanulás szempontjából. Tegyük fel, hogy egy cég telefonos, szövegfüggetlen, nyílt halmazú beszélőfelismerést szeretne abból a célból, hogy a betelefonáló kliensek adatait rövid hangminta után a rendszer automatikusan kilistázza az operátornak. Egy nagyobb telefontársaság esetén ez több ezer beszélőt is jelenthet. Hagyományos neurális hálózatokkal minden egyes klienstől több perces hangmintára lenne szükség, hogy ne tanítsuk túl a modellt.

A másik, még nagyobb probléma, hogy új, vagy távozó kliens esetén újra kell tanítani a teljes hálózatot. A few-shot learning tökéletesen megoldja mindkét problémát. Kevés tanítómintával – regisztrációkor pár kérdés kliensenként – működik a rendszer és rugalmas is tanítás szempontjából; új kliens esetén hozzáadjuk a segédhalmazhoz a hangmintáját, távozáskor pedig eltávolítjuk.

4.2. Metrikus metatanítás

Metrikus metatanítás során a modell a tanítóminták halmazán egy távolságfüggvényt tanul meg.

$$P_\theta(y|x, S) = \sum_{(x_i,y_i)\in S} k_\theta(x, x_i)y_i \tag{4.1}$$

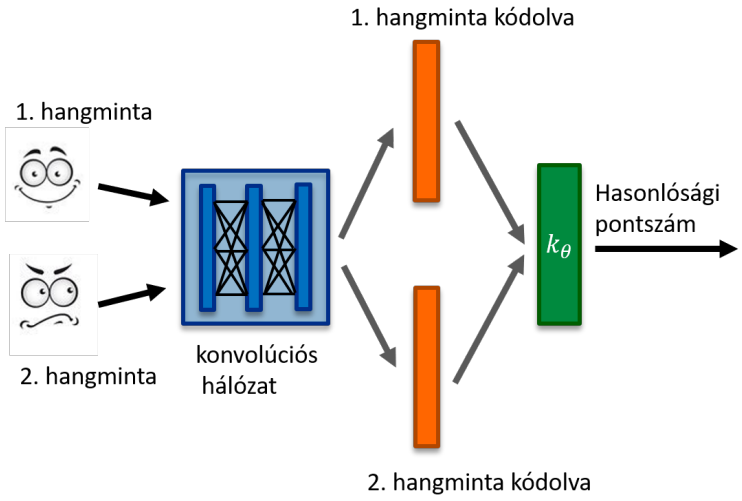
A valószínűsége annak, hogy a θ modellparaméterekkel az x minta S segédhalmaz mellett az y osztályba tartozik egyenlő a segédhalmazba tartozó címkék súlyozott összegével. A súlyt a k_θ ún. kernel függvény számolja ki. A kernel függvény az a távolságfüggvény, amit a modell megtanul [23].

A megközelítésre több implementáció is létezik, mint például a Matching Network, Prototypical Network, Relation Network és a konvolúciós sziámi hálózat. Ezek közül az utolsót fogom részletesen bemutatni.

4.2.1. Konvolúciós sziámi hálózatok

A konvolúciós sziámi hálózatok két konvolúciós ikerhálózatból épülnek fel, amelyek megosztott súlyokat és rétegeket használnak. A hálózat két bemenetet kap, és miután a konvolúciós rétegek kiszámolták a jellemző vektorokat, ezeknek egy θ távolságfüggvénnyel

méri a hasonlóságát [24]. Ha a hasonlósági pontszám meghalad egy küszöbértéket, a két hangminta egy beszélőtől származónak tekintjük, egyébként különbözőnek, de ez már a felhasználáshoz tartozik.



4.1. ábra. Konvolúciós sziámi hálózat architektúrája.

Sok ábrán a sziámi hálózatokat két azonos hálózattal reprezentálják. Valójában mivel a két ikerhálózat súlyai és rétegei megosztottak, elég egy hálózatot használni és csak a jellemző vektorokat eltárolni, így kevesebb erőforrást használunk.

Működése közben egy mintáról nem azt tanulja meg, hogy melyik osztályba tartozik, hanem a különbséget a többi mintához képest. Tanítás közben a konvolúciós hálózat súlyait javítjuk, hogy az általa képzett kódolások a minták hasonlósága alapján a metrikus tér megfelelő pontjaira képezze le azokat. Ez azt jelenti, hogy ha a hálózat a hangokból 10 dimenziós hangvektorokat képez; az azonos beszélőtől származó hangmintákat egymáshoz közel, a különbözőket egymástól távol helyezi el. A távolságot mindig a távolságfüggvény alapján számoljuk ki.

A konvolúciós sziámi hálózatok megoldást jelentenek a few-shot learning problémára, mert kevés tanítómintával is jól működnek. Vegyük példaként egy vállalat beszélőazonosító rendszerét. A sziámi hálózatnak elég egy segédhalmazban egy-egy képet eltárolni minden alkalmazottról. Amikor egy alkalmazott a rendszert használja, a képét összehasonlítja a segédhalmazban lévő képekkel és a hasonlóság alapján dönt. Ezzel két problémát old meg:

1. Egyrészt nem szükséges minden alkalmazottról sok képet készíteni.
2. Másrészt új alkalmazottak érkezése, vagy egy alkalmazott távozása után nem szükséges újra tanítani a hálózatot.

Fontos megjegyezni, hogy itt nem arról van szó, hogy a modellt egyáltalán nem tanítjuk. A beszélőfelismerő modellt tanítani kell sok beszélővel és mintával, azért, hogy megtanulja a vektorok leképzését. De utána már felhasználható lesz eddig nem látott beszélők azonosítására is csupán a segédhalmazban megadott mintákat használva, tehát egy metatanuló hálozatnak tekinthető, ami definíció szerint megoldja az *n-way-k-shot* feladatot.

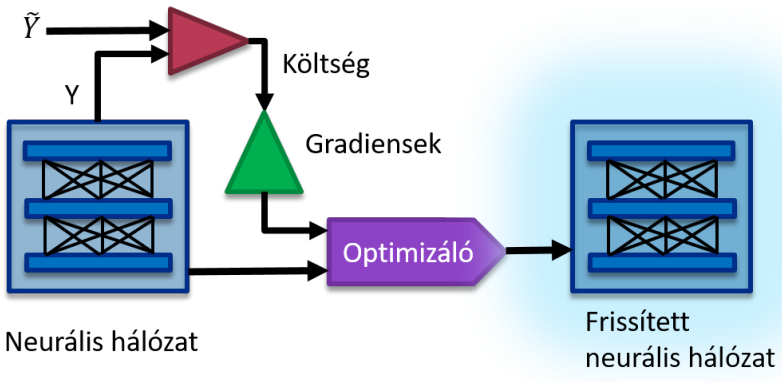
4.3. Optimizációs metatanítás

Az optimizációs algoritmusok mindig egy célfüggvényt szélsőértékét keresik. Neurális hálózatokban ennek a célfüggvénynek a költségfüggvény felel meg és függ a modell tanulható paramétereitől. A optimizációs metatanítási megközelítések a modellek egyes paramétereit szintén modelleknek tekintik. Ezek a paraméterek tipikusan a modell kezdeti paraméterei (súlyok, eltolássúly) és az optimizációs algoritmus [23].

A fejezetben bemutatok egy megközelítést, ahol az optimizációs algoritmust modellezik az adott feladat tanításának felgyorsítása érdekében és két másik few-shot learning megoldást: a MAML és Reptile algoritmusokat, ahol a modell kezdeti paramétereit állítják be úgy, hogy könnyen adaptálható legyen új feladatokhoz.

4.3.1. Optimizáló algoritmus modellezése

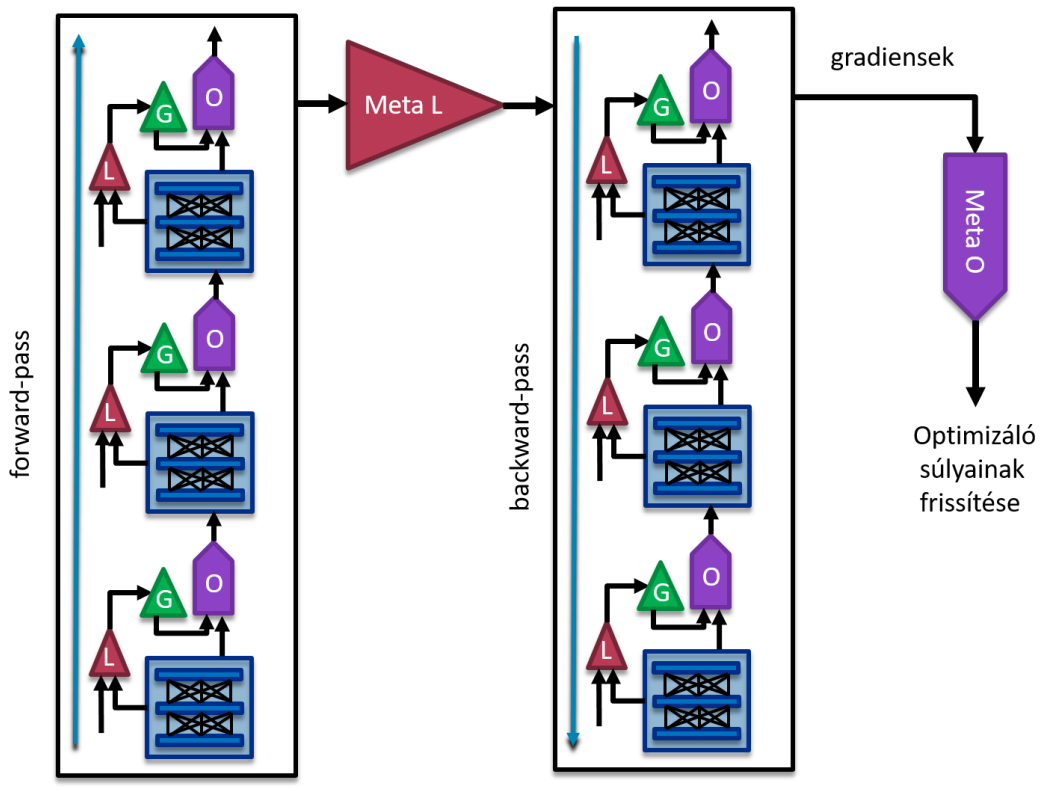
Egy neurális hálózat tanulását mutatja a 4.2 ábra. A neurális hálózat által számolt kimenet és az elvárt kimenet közötti különbség alapján a költségfüggvénnyel számoljuk ki a hibát. Ezt jelöli a piros háromszög. A zöld háromszög a költségfüggvény gradienseit számolja ki az egyes rétegekre. Az optimizáló megkapja a gradienseket és a neurális hálózat súlyait, majd javít rajtuk [25].



4.2. ábra. Neurális hálózat architektúra.

Tegyük fel, hogy a 4.2 ábrán látható neurális hálózatot bináris klasszifikációra használjuk és hívjuk simán modellnek. A modell kezdeti paramétereit minden iterációban az optimizáló frissíti. A metatanítás az optimizációs algoritmus szempontjából azt jelenti, hogy az optimizáló állítható paramétereit nem mi állítjuk be kézzel, hanem a feladatot átadjuk egy másik modellnek, amit megtanítunk arra, hogy ezeket optimizálja miközben az eredeti modellünk tanul. Tehát az optimizáló az eredeti modellünk súlyait állítja, miközben a másik modellünk az optimizáló paramétereit javítja.

Ezzel absztrakciós szintet léptünk, ezt a másik modellt nevezzük metamodellnek. A metamodellnek ugyanúgy lesz meta-költségfüggvénye, meta-gradiensei és meta-optimizálója. Ugyanakkor látnunk kell, hogy ezt a metaoptimizálót is tekinthetjük modellnek és léphetünk feljebb metaszinteket, de előbb utóbb szükség lesz egy konkrét meta-optimizálóra, ami az alatta lévő optimizáló paramétereit állítja.



4.3. ábra. Optimizáló metatanítás.

A 4.3 ábra az optimizáló metatanítást szemlélteti. A konkrét modell szinten a fekete téglalapokban függőlegesen az eredeti bináris klasszifikációra használt modellünk tanítása történik, míg a téglalapok között vízszintesen a metamodell tanítása látszik.

A metaköltségnek vehetjük az eredeti modell költségeinek összegét egy adott iterációig. Ez jól leírja, hogy a modell tanul-e. A metaköltség-függvény alapján kiszámoljuk a meta-gradienseket, amit átadunk a metaoptimizálónak. Ez már egy konkrét optimizáló, például ADAM. Ezután a meta-optimizáló állítja az optimizáló paramétereit úgy, hogy csökkentse a meta-költséget, vagyis javítja a tanulási folyamatot.

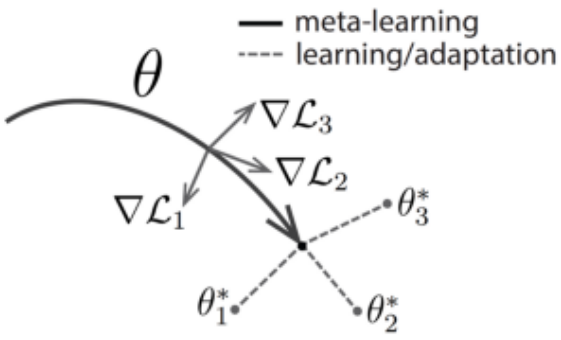
4.3.2. Model-Agnostic Meta Learning (MAML)

A MAML a modell kezdeti paramétereit optimalizálja úgy, hogy gyorsan tanuljon. Célja, hogy új, hasonló feladatokra kevesebb – akár egy – iterációval jó eredményt érjen el az optimizációs algoritmus. A modellt több feladathoz adaptálja a kezdeti paraméterei beállításával, így az kevés gradiens frissítés után képes megtanulni új feladatokat. Ez egy megközelítés a few-shot learning problémára [26].

Legyen a modell egy f_θ függvény, ahol θ jelöli a modell paramétereit. Amikor a modellt egy új $\mathcal{T}_i$ feladathoz adaptáljuk, a modell θ paramétereit változtatjuk θ_i'-re. Az adaptált paraméter egy gradiens frissítés esetén a következő:

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta) \tag{4.2}$$

A 4.4 ábrán a θ jelöli a modell kezdeti súlyait. A szürke vonalak a $\nabla\mathcal{L}_1$, $\nabla\mathcal{L}_2$, $\nabla\mathcal{L}_3$ gradi-



4.4. ábra. A MAML algoritmus vizualizációja. (Finn et al 2017)

enseket mutatják, a θ_1^*, θ_2^*, θ_3^* pedig az adott feladathoz adaptált modell optimális paraméterei. A vastag vonal a metatanulási folyamatot mutatja. Látható, hogy a θ paraméterű modell jelenlegi helyzetében közel van mindhárom feladat optimális paramétereihez, tehát kevés gradiens frissítéssel finomhangolható bármelyikre.

Algorithm 1 Model-Agnostic Meta Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β step size hyperpatameters
1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to K examples
6: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
7: **end for**
8: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
9: **end while**

Az algoritmus először veszi a $\mathcal{T}$ feladatok egy $p(\mathcal{T})$ eloszlását és a modell kezdeti paramétereit véletlen módon inicializálja. A feladatok közül kiválaszt párat és mindegyikre K tanítómintával tanítja a modellt. A tanítás során gradiens frissítésekkel kiszámolja az optimális θ_i modellparamétereket 4.2 szerint. A meta-célfüggvényt az adaptált θ_i paraméterekkel kiszámolt költségek összege adja a $\mathcal{T}_i$ feladatokon.

$$\min_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'}) = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)}) \tag{4.3}$$

Mielőtt új $\mathcal{T}_\rangle$ feladatokat választ, sztochasztikus gradient descent módszerrel frissíti a modell kezdeti paramétereit a meta-célfüggvény szerint.

4.3.3. Reptile

A reptile algoritmus nagyon hasonlít a MAML-hoz. Ugyanúgy a hálózat kezdeti súlyait incicializálja úgy, hogy további hasonló feladatokra könnyen általánosítható legyen. A MAML-hoz képest előnye, hogy kevesebb számítást igényel, nincs szükség második deriváltak kiszámolására, csak SGD-t futtat a feladatokon [27].

Adott a szigma kezdeti paraméter vektor. Valamennyi iteráción keresztül választunk egy

Algorithm 2 Reptile

Initialize ϕ, the vector of initial parameters
for iteration = 1, 2, ... **do**
    Sample task τ, corresponding to loss L_τ on weight vectors $\tilde{\phi}$
    Compute $\tilde{\phi} = U_\tau^k(\phi)$, denoting k steps of SGD or Adam
    Update $\phi \leftarrow \phi + \epsilon(\tilde{\phi} - \phi)$
end for

véletlen feladatot és futtatjuk rajta valahányszor az SGD algoritmust, ami a ϕ paraméter vektorból a $\tilde{\phi}$-t eredményezi. Ezután javítjuk a modell kezdeti paramétereit a megadott szabály szerint.

4.4. Modell alapú metatanítás

A modell alapú metatanítás lényege az olyan modelleket tervezése, amelyek képesek kevés tanítással, gyorsan javítani a paramétereiket. Egy ismert implementáció a Memory-Augmented Neural Networks, ami neurális Turing-gépeket használ [23].

5. fejezet

Android alkalmazás beszélőfelismerésre

A következő fejezetben bemutatom egy általam a beszélőfelismerés szemléltetésére készített android alkalmazás fő elemeit és működését, a kapcsolódó technológiákat és implementációs részleteket. Az alkalmazásban konkrét beszélőfelismerésre készített neurális hálózatokat használok fel. A cél nem egy új beszélőfelismerő modell készítése volt, hanem az eddigiek felhasználásával egy működő alkalmazás létrehozása.

Az alkalmazás kliens-szerver architektúrájú. Az android kliens segítségével a felhasználó regisztrálhat a rendszerbe névvel és hanggal, majd az azonosítás opció megnyomásával hangmintát adva az alkalmazás eldönti, hogy regisztrálva van-e, és ha igen, akkor visszaadja az illető nevét.

5.1. Modell predikció a felhőben vagy lokálisan?

A szervernek tárolnia kell a felhasználóktól származó hangmintákat és a felhasználó azonosítás folyamata közben összehasonlítani őket. Az összehasonlítást egy sziámi neurális hálózat végzi. Az alkalmazás felépítését tekintve a legfontosabb kérdés, hogy ezt az összehasonlítást központilag egy felhőben a szerver, vagy a kliens oldali alkalmazás végezze. Mivel a modell végzi a predikciót, ettől a döntéstől függően azt a szerveren vagy a kliens eszközökön kell tárolni. Mindkét architektúrának vannak előnyei és hátrányai is.

Mivel esetünkben az alkalmazás kész, előre tanított modelleket használ, nincs szükség a modellek tanítására. Ennek ellenére ha a jövőben saját modellt szeretnénk használni felmerül a kérdés, hogy hol tanítsuk azt [28].

- A felhőben hosztolt gépi tanulási szolgáltatások általában saját modelleket használnak. Mi a saját adatainkat átadjuk és a szolgáltatás gondoskodik a modell tanításáról. Ezután a predikciót egy API-n keresztül végezhetjük el. Figyelni kell arra, hogy ebben az esetben nem mindig miénk a modell. Ha nincs lehetőség tanítás után a modell letöltésére, akkor a predikció mindenképp a felhőben marad. Másrészről a magas szintű szolgáltatások kezelése könnyebb, nem kell érteni a neurális hálózatok tanításához, de nem elég flexibilisek. Általában nem változtathatunk a modellen és az API-n sem.
- A felhőben taníthatjuk a modellünket az erre készített szolgáltatások nélkül is nagyobb hozzáértést feltételezve. Ekkor megválaszthatjuk a modell típusát és a technológiákat (Tensorflow, Pytorch, stb.) és teljes mértékben miénk a modell és az irányí-

tás. A lokális tanítással szemben további előny, hogy az erőforrásokat rugalmasan fel-le skálázhatjuk.

- Lokálisan is taníthatjuk a modellt. Ez főleg akkor éri meg, ha van elég erőforrásunk hozzá vagy a modell mérete nem indokolja több erőforrás használatát. Nagyobb modellek esetében a tanítási idő hosszú. Az árakat és az időt mérlegelve dönteni kell, hogy a lehetőségek közül melyiket indokolt választani.

Ha már rendelkezünk egy működő modellel, akkor el kell döntenünk, hogy a predikciót a felhőben vagy lokálisan az eszközön végezzük.

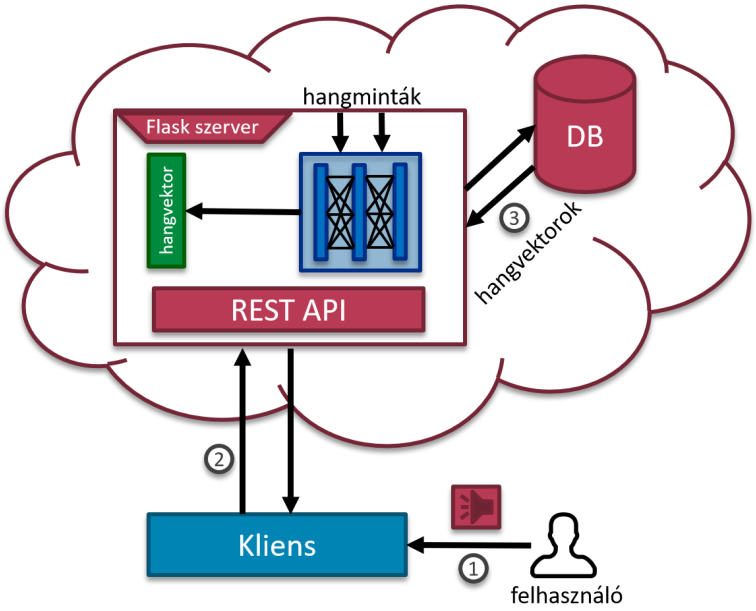
- A hálózati kapcsolatot tekintve ha a predikciót lokálisan végezzük, nincs szükség internetkapcsolatra, predikció a készüléken történik. Ellenkező esetben a kliens interneten keresztül kérést küldd a szervernek, ami elvégzi a predikciót és visszaküldi a választ.
- A modellt felhőben a szerveren tárolni biztonságosabb. Lokális predikció esetén a modellt a készüléken tároljuk. Ilyenkor gondoskodni kell a biztonságáról, hogy ne lehessen visszafejteni azt (reverse engineering).
- A predikció sebessége is meghatározó szempont. Ha lokálisan, a mobiltelefon hajtja végre, a felhőhöz képest kisebb erőforrással gazdálkodunk, ami miatt lassabb lesz a számítás. Viszont ha a felhőben végezzük, a kommunikációs overhead hozzáadódik a válaszhoz. Le kell mérni, hogy a kliens-szerver kommunikáció késleltetése mekkora az erőforráskülönbségből származó számítási időkhöz képest.
- Az alkalmazás architektúráját tekintve ha a modell és a predikció a szerveren történik, bármikor frissíthetjük, finomhangolhatjuk a modellt és ehhez nem kell a felhasználóknak frissítéseket letölteniük. Nincs szükség szinkronizációra. Az alkalmazás elosztott, elkülönült backend és frontend részekből áll, amelyek egy interfészen keresztül kommunikálnak, így ezek a komponensek lecserélhetők, csak az interfészt kell implementálniuk (API).
- Felhőbeli erőforrásokat használva az árat is figyelembe kell venni. Minél több felhasználó használ egy alkalmazást és amennyiben a számítás központilag a felhőben történik, annál több erőforrásra van szükség. Nagy előnye a készüléken végzett predikciónak, hogy jól skálázódik. Több felhasználó esetén nincs szükség több erőforrásra.

Jelen alkalmazás egy beszélőfelismerő szoftvert szemléltet, amelyet a valóságban vállalatok üzemeltetnek beléptető kapuval biometrikus azonosításra használva. Központi szerverre szükség van a regisztrált felhasználók adatainak biztonságos tárolása és a könnyű szinkronizáció miatt. Ha feltesszük, hogy a beléptető rendszert nem több mint tízezer ember azonosítására használják és a belépés szekvenciálisan történik, óriási erőforrásokat sem kell használni, mert a szerver egy időben legfeljebb a beléptető kapuk száma szerinti számítást kell végezzen, ami korlátos. Ezeket figyelembe véve úgy döntöttem, hogy a modell tárolása és a predikció szerveren fog történni.

5.2. Alkalmazás architektúra és általános működés

Mivel az alkalmazás tárolja a regisztrált felhasználók hangmintáit és a modellt; a biztonság, a könnyű központi adminisztráció és a tény, hogy nincs szükség szinkronizációra

mind megerősíti a kliens-szerver architektúra szükségességét és előnyeit, így peer-to-peer megoldás fel sem merült.



5.1. ábra. Beszélőfelismerő alkalmazás: Regisztrációs fázis.

A 5.1 ábrán látható kliens a felhőben futó szerverrel egy REST API-n keresztül kommunikál. A kliens segítségével a felhasználó regisztrál a rendszerbe, ezután azonosíthatja magát. A regisztrációs fázis működése a következő:

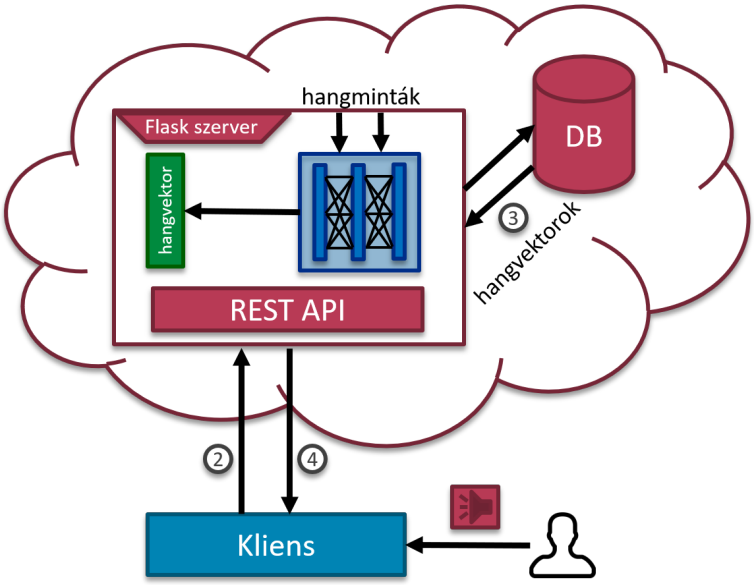
1. A kliens alkalmazás rögzíti a felhasználó hangját és nevét.
2. A kliens a hangmintát és a nevet elküldi a szervernek a REST API-n keresztül.
3. A szerver előfeldolgozza a hangfájlt, a modell segítségével hangvektort készít belőle és menti az adatbázisba.

Miután a felhasználók regisztráltak, az alkalmazás a hangjuk alapján azonosítani tudja őket. Az azonosítási fázis lépéseit a 5.2 ábra mutatja.

1. A kliens rögzíti az azonosítani kívánt felhasználó hangját.
2. A hangmintát továbbítja a szervernek.
3. A szerveren futó alkalmazás a hangfájlt előfeldolgozza és hangvektort készít belőle.
4. A szerver kiszámolja a hangminták közötti hasonlósági pontokat. A minimálishoz tartozó felhasználó nevét elküldi a kliensnek.

5.2.1. Biztonsági funkciók

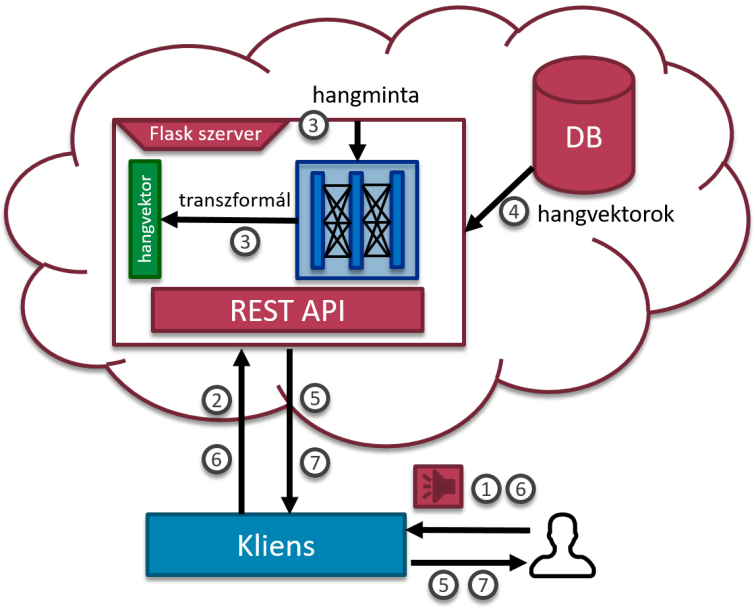
Azt is figyelembe kell venni, hogy azonosításkor a szerver a minimális vektortávolsághoz tartozó névvel tér vissza. Ez azt jelenti, hogy ha egy regisztrált felhasználó van a rendszerben, bárki képes lenne belépni a regisztrált felhasználó nevében, hiszen csak egy távolság van, ami minimális. Ennek elkerülésére további biztonsági funkciókkal kell



5.2. ábra. Beszélőfelismerő alkalmazás: Regisztrációs fázis.

kiegészíteni az alkalmazást.

Meg kell határozni egy biztonsági küszöböt a vektorok távolságára - azaz ha nem elég hasonlóak, egyéb autentikációs mechanizmusra lesz szükség. Ennek megfelelően a szervert kiegészítettem egy biztonsági küszöbbel, amit ha azonosításkor a vektortávolság meghalad, jelszó alapú azonosítást kér a felhasználótól. A biztonsági funkcióval kiegészített működés abban az esetben, ha a biztonsági küszöböt meghaladja az alkalmazás a 5.3 ábrán látható. A regisztrációs fázis annyiban tér el az előzőtől, hogy a felhasználó egy jelszót is megad a neve mellett, amit a szerver eltárol.



5.3. ábra. Beszélőfelismerő alkalmazás: Azonosítás jelszó alapú autentikációval.

1. A kliens rögzíti az azonosítani kívánt felhasználó hangját.
2. A hangmintát továbbítja a szervernek.
3. A szerveren futó alkalmazás a hangfájlt előfeldolgozza és hangvektort készít belőle.
4. A szerver összehasonlítja a hangvektort a korábban eltároltakkal és kiszámol egy hasonlósági pontot vagy vektortávolságot.
5. A pontszám meghaladja a biztonsági küszöböt, ezért kliensnek felszólítást küld, hogy jelszó alapú autentikációt kér.
6. A kliens megadja a saját jelszavát, amivel korábban regisztrált és elküldi a szervernek.
7. A szerver ellenőrzi, hogy a megadott jelszó valóban az azonosítani kívánt felhasználóhoz tartozik-e. Amennyiben igen, visszaküldi az azonosított felhasználó nevét, egyébként jelzi, hogy az autentikáció sikertelen volt.

5.2.2. Vektorok átlagolása

A vektorok átlagolása egy opcionális funkció. A szerver oldali alkalmazás minden sikeres azonosítás után átlagolja a tárolt vektort az aktuálisan azonosítottal. Azonosításkor ha a hasonlósági pontszám a biztonsági küszöb alatt van, az átlagolás minden további nélkül megtörténik. Amennyiben a minimális vektortávolság nem üti meg a küszöbszintet az alkalmazás az aktuális vektort ideiglenesen eltárolja. Ezután ha a felhasználó sikeresen autentikál jelszóval, a vektort átlagolja a korábbival majd törli, egyébként csak törli azt.

5.2.3. Konfiguráció

A szerver oldali alkalmazás képes többféle hasonlósági pontot számolni a vektorok között. A jelenlegi implementáció a következő hasonlósági pontokat támogatja $v_1 = (x_1, x_2, \dots)$ és $v_2 = (y_1, y_2, \dots)$.

- Euklideszi-távolság:

$$d_{euc}(v_1, v_2) = \left(\sum_{i=1}^{n} x_i - y_i \right)^{\frac{1}{2}}$$

- Koszinusz-távolság:

$$d_{cos}(v_1, v_2) = \frac{v_1 \cdot v_2}{\|v_1\| \cdot \|v_2\|} = \frac{\sum_{i=1}^{n} x_i \cdot y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \cdot \sqrt{\sum_{i=1}^{n} y_i^2}}$$

Konfigurálható továbbá a biztonsági küszöbszám is. Ez annak az értéke, amelyet ha a hasonlósági pontszám meghalad, a szerver jelszó alapú autentikációs kérést küld a kliens fele.

5.3. Implementáció

5.3.1. Kliens alkalmazás

Ebben a fejezetben részletesen bemutatom az alkalmazás fő komponenseinek működését, felépítését és a fejlesztőkörnyezet részeit.

5.3.1.1. Projekt felépítése

Az Android Studios projekt főbb elemei a következők:

- *AndroidManifest.xml*: Engedélyek és activityk deklarálása.
- *java*: Activitykhez tartozó és egyéb java osztályok.
- *res/layout*: A felhasználói felülethez tartozó xml leírók.
- *build.gradle*: Gradle modul konfiguráció és dependenciák módosítására.

5.3.1.2. Engedélyek

Bizonyos műveletek elvégzéséhez az Android operációs rendszeren engedélyek szükségesek. Az engedélyek a művelet típusától függően *normál* vagy *veszélyes* osztályba tartoznak. Ha az alkalmazás olyan műveletet akar végrehajtani, amihez engedély szükséges, azt deklarálni kell az android projekt manifest fájljában.

A *normál* engedélyeket az operációs rendszer ezután automatikusan biztosítja, míg a *veszélyesekhez* a felhasználó engedélyét kéri. Korábbi android verziókban az engedélyek megadási az alkalmazás telepítésekor történt, de később ez megváltozott. A minimum Android 6.0-át (23-as API szint) használó készülékeken a felhasználó már nem telepítéskor, hanem futási időben kell megadja az engedélyeket. Egy felugró ablak listázza, hogy az alkalmazás milyen engedélyeket kér, ezután választhat, hogy melyeket adja meg neki.

A beszélőfelismerő alkalmazás a következő engedélyeket használja:

- RECORD_AUDIO: A mikrofon használata hangfelvételhez.
- READ_EXTERNAL_STORAGE: A külső tárhely olvasásához.
- WRITE_EXTERNAL_STORAGE: A külső tárhely írásához.
- INTERNET: Internetelérés hálózati kommunikációhoz.

5.3.1.3. Tárhely

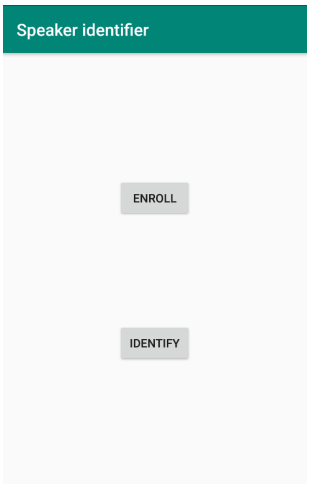
Az alkalmazás a felhasználótól vett hangmintát *wav* formátumban tárolja el. A *wav* egy tömörítetlen audioformátum és mivel később hangmintából jellemzőkinyerés történik, így nem veszíthetünk információt a jellemzőkről a tömörítés miatt. Később bemutatom, hogy a tömörített audioállományok milyen hatással vannak a modell teljesítményére.

Az Android beépített *MediaRecorder* osztálya a *wav* formátumot nem támogatja és nincs erre készült más beépített Java osztály sem, ezért egy erre készített kódot használtam fel Selvaline blogjáról. Selvaline *WavRecorder* osztálya az általunk beállított paraméterekkel (csatorna, mintavételi frekvencia, stb.) felveszi a hangot és egy *wav* fájlban tárolja azt.

Az Android operációs rendszeren használhatunk külső és belső tárhelyet. A belső tárhely előnye, hogy a fájlokhoz csak az alkalmazás fér hozzá és annak eltávolításakor a hozzá kapcsolódó fájlok is törlődnek. A külső tárhely előnye esetünkben, hogy könnyű mountolni. Ha a készüléket USB-vel számítógéphez csatlakoztatjuk, a külső tárhelyen lévő fájlokat böngészhetjük. Ez a hangfájl tesztelése miatt célszerű választás volt.

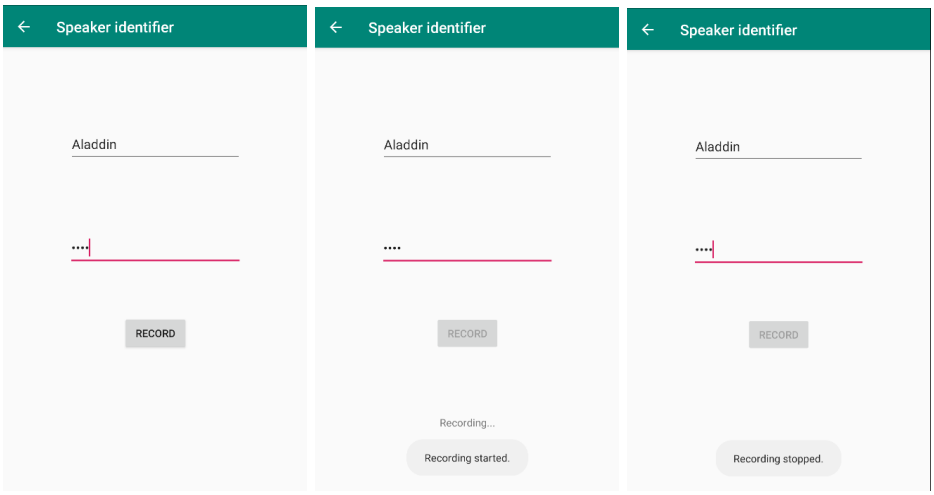
5.3.1.4. Felhasználói felület

A felhasználói felület három részből áll és minden felülethez egy-egy *Activity* osztály tartozik. A fő felületen választhat a felhasználó a regisztráció és az azonosítás között. A fő felületen az *Enroll* gombot megnyomva a regisztrációs, illetve ehhez hasonlóan az *Identify* opcióval az azonosító felületre ugorhatunk.



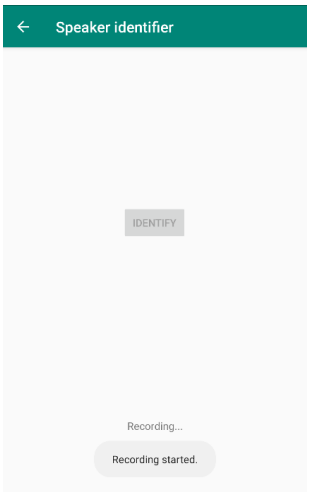
5.4. ábra. Menü.

A regisztrációs felületen megadjuk a nevet, majd a *Record* gombra kattintva három másodperc múlva elindul a hangfelvétel, ami négy másodpercen keresztül tart. A felületen megjelenő információ folyamatosan tájékoztatja a felhasználót a hangfelvétel folyamatáról.



5.5. ábra. Felhasználói felület regisztrációhoz.

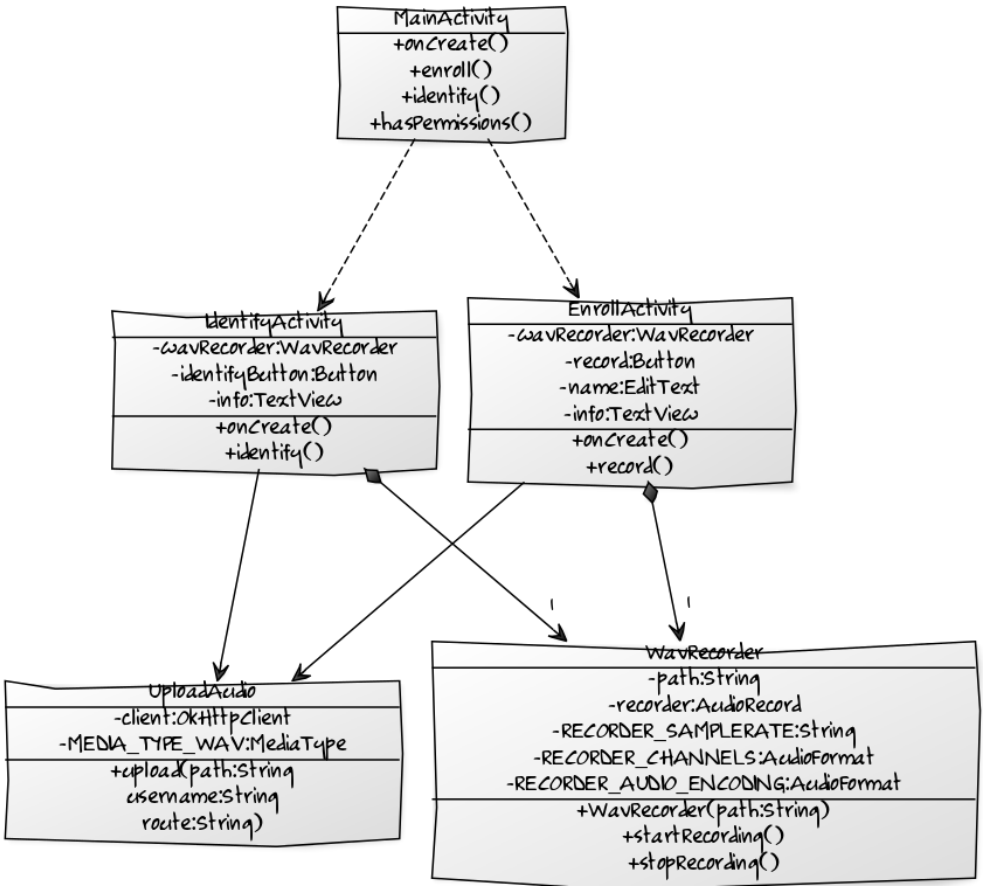
A zöld fejlécen a vissza nyílra kattintva visszajutunk a főmenübe. Ezt az opciót a manifest fájlban a *parentActivityName* tag megadásával érhetjük el.



5.6. ábra. Felhasználói felület azonosításhoz.

A főmenüből az azonosító felületre lépve az *Identify* gombra kattintva szintén visszaszámlálásra elindul a hangrögzítés, ami négy másodperig tart. Ezután megkapjuk a választ; az azonosított felhasználó nevét, vagy a nem sikerült azonosítani üzenetet.

5.3.1.5. Osztálydiagram és részletes működés



5.7. ábra. Beszélőfelismerő alkalmazás: Osztálydiagram.

A *MainActivity* osztály tartozik a főmenühöz. Amikor a főmenübe lépünk, az *onCreate* metódus hívódik meg, amely ellenőrzi, hogy az alkalmazásnak megvannak-e a szükséges engedélyei a *hasPermission* segédmetódus által. Ha igen akkor nem tesz semmit, egyébként a felhasználótól az összes engedély megadását kéri.

Az *enroll* metódus a felületen az *Enroll* gombhoz, az *identify* az *Identify* gombhoz tartozik. Mindkét függvény létrehoz egy-egy *Intent*-et ami a hivatalos definíció szerint egy elvégzendő művelet absztrakt leírása. Az *Intent* valójában arra jó, hogy egy felületről egy másik felületre lépjünk. A kiinduló felületen egy gombra kattintva (pl. *Enroll*), a gombhoz tartozó listener metódus (*enroll()*) hívódik meg, ami létrehoz egy *Intentet* paraméterként átadva az elérni kívánt felülethez tartozó *Activity* osztályt (*EnrollActivity*). Ezután az *Intentet* elindítva átlépünk a felületre.

A regisztrációs felületre lépve az *onCreate* metódus meghívásakor az *EnrollActivity* példányosít egy *WavRecorder* objektumot a fájl elérési útjával paraméterezve. A *record* metódus visszaszámlál, kiírja a felhasználónak az időzített üzeneteket, és elindítja majd megállítja a *WavRecordert*.

Az időzítést az ajánlás szerint a *Handler* osztály segítségével végeztem. Ennek előnye, hogy az időzített feladatokat egy háttérszálon futtatja, így nem blokkol. Így nem történhet meg, hogy egy hosszabb feladat elvégzése miatt a felületen nem kattinthatunk amíg az nincs kész.

Az *IdentifyActivity* osztály nagyon hasonlóan működik. A különbség, hogy a *EnrollActivity*-hez képest nem küld nevet és más címre küldi a hangfájlt.

Mindkét osztály a *UploadAudio* objektumot használ a fájl szerverhez való elküldéséhez. Az *UploadAudio* egy *OkHttpClient* http klienst használ és *multipart/form* Http POST kérést küld a szerver felé a hangfájllal ill. regisztrációkor a névvel. Ezután egy callback metódussal várja a szerver válaszát, amit az azonosító felületre továbbít.

5.3.2. Szerver

Szerver oldalon egy Flask webalkalmazás fut felhőben és REST API-val rendelkezik. Fogadja a klienstől érkező hangfájlokat, előfeldolgozza és tárolja őket. Azonosításkor a modellen prediktál, majd visszaküldi az eredményt a kliensnek.

5.3.2.1. Flask

A Flask egy Python nyelvhez készült web mikrokeretrendszer. Mikrokeretrendszernek nevezzük a minimális webalkalmazás keretrendszereket, amelyekből általában hiányoznak a autentikációs, autorizációs könyvtárak, az objektum-relációs leképzés stb.

A Flask két fő BSD-licencelt komponensre épül:

- Werkzeug: A Werkzeug egy Python toolkit WSGI alkalmazásokhoz. Keretrendszereket lehet építeni rá és többféle Python verziót támogat. A WSGI egy hívási konvenció Pythonban írt webalkalmazásokhoz történő kérésekre.
- Jinja: Web template engine Python alkalmazásokhoz.

A Flask előnye, hogy nagyon gyorsan és könnyedén lehet vele REST API-t készíteni és

elindítani egy webalkalamzást. Nincs szükség más könyvtárakra, toolokra hozzá.

Képes egyszerre több kérést is kezelni. Külön sessionökből érkező kérések külön szálakon futnak, így a keretrendszer a szálkezelést is megoldja helyettünk. A következő fontosabb funkciókat biztosítja:

- Webszerver fejlesztésre és debugger
- Unit tesztek integrált támogatása
- REST támogatása
- Jinja2 template weboldalakhoz
- WSGI 1.0 szabvány
- Jól dokumentált

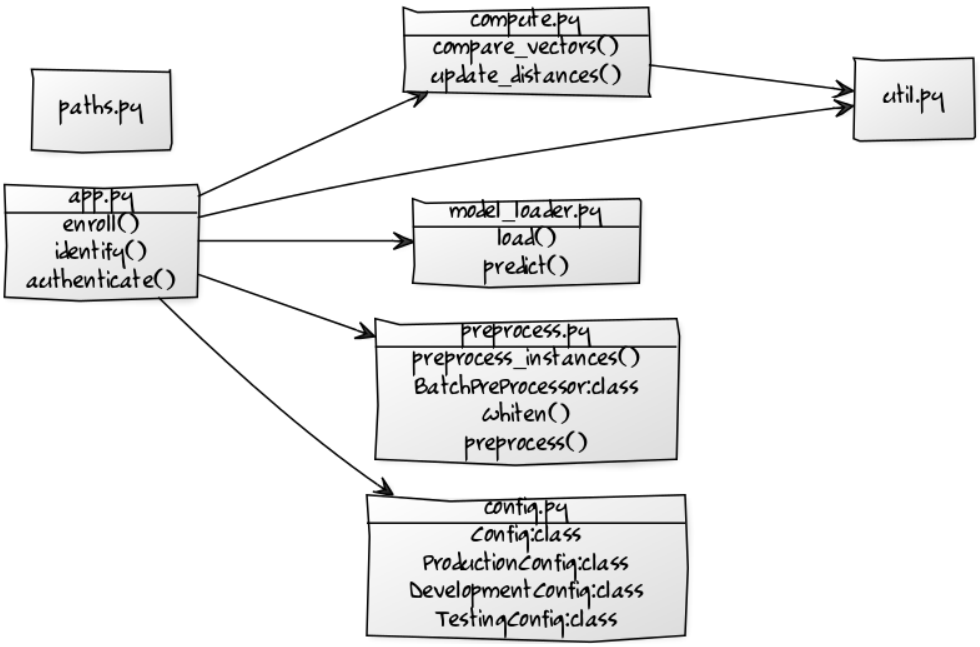
5.3.2.2. Projekt felépítése

- *app*: A webalkalmazást indító és a logikát tartalmazó (előfeldolgozás, predikció, fájlok kezelése) illetve konfigurációt kezelő Python szkriptek könyvtára (app.py, preprocess.py).
  - app/app.py
  - app/model_loader.py
  - app/preprocess.py
  - app/config.py
  - app/paths.py
- *data*: Tartalmazza a klienstől regisztrációs fázisban kiszámolt vektorokat, az átlagolásra váró vektorokat és felhasználói adatokat.
  - data/vectors: Az előfeldolgozott hangfájlok.
  - data/vectors_to_average: Az átlagolásra váró, ideiglenesen tárolt vektorok helye.
  - user_dict: Felhasználó UID - felhasználónév összerendeléseket tartalmazó dictionary.
  - passwords: Felhasználó UID - jelszó párokat tartalmazó dictionary.
- *model*: A beszélőfelismerésre használt modelleket tároló mappa.

5.3.2.3. Részletes működés

Az alkalmazás a következő REST végpontokat biztosítja a kliens számára:

- /enroll: A regisztráció végpontja.
- /identify: Az azonosítási végpont.
- /authenticate: A jelszó alapú autentikáció végpontja.



5.8. ábra. Flask szerver: Osztálydiagram.

- *app.py*: Az *app.py* implementálja a REST végpontokat. Az *app.py* először létrehoz egy Flask objektumot, ezután betölti a megfelelő konfigurációt a *config.py* szkriptben deklarált valamelyik objektummal. Betölti a modellt, majd elindítja az alkalmazást, ami a */enroll*, */identify* és */authenticate* útvonalakon *HTTP POST* kérésekre vár.

  Mivel az alkalmazás egy prototípus és nincs szükség felhasználóspecifikus adatok tárolására néven és jelszón kívül, adatbázis helyett *dictionary* tárolja az egyedi azonosító - felhasználónév illetve egyedi azonosító - jelszó párokat egy fájlba szerializálva, a hangvektorok pedig egyedi azonosítónévvel vannak eltárolva.

  – *enroll()*: Regisztrációkor megkapja a klienstől a hangmintát, a nevet és a jelszót. Amennyiben a hangminta nem támogatott formátumú, HTTP 400 (Bad Request) státuszkóddal válaszol.

    Egyébként a felhasználónak létrehoz egy egyedi azonosítót és elmenti a jelszavát az *passwords* dictionarybe. Preprocesszálja a hangmintát, a modell segítségével hangvektort készít belőle majd eltárolja.

    A hangvektorok bináris formátumban kerülnek mentésre egyedi azonosító néven, így könnyen visszakereshetők.

    Végül frissíti a *user_dict* dictionaryt az aktuális felhasználó nevével és azonosítójával.

  – *identify()*: A kliens ide küldi a hangmintát azonosításra. Amennyiben a hangminta nem támogatott formátumú, HTTP 400 (Bad Request) státuszkóddal válaszol. Ha a hangminta támogatott preprocesszálja, majd hangvektort készít belőle. Összehasonlítja a korábban regisztrált felhasználók hangvektorjaival, és megkeresi a minimális hasonlósági pontszámot.

    Ha a minimális távolság meghaladja a biztonsági küszöböt, az aktuális

vektort menti későbbi lehetséges átlagolás céljából. Ezután jelszó alapú autentikációs kérést küldd a kliensnek HTTP 403 (Forbidden) státuszkóddal. Mivel a HTTP protokoll állapotmentes, a kérés fejlécében elküldi a minimális távolsághoz tartozó felhasználó azonosítóját is az egyszerűség kedvéért (session kezelés helyett).

Ha a minimális távolság a biztonsági küszöbön belül van, azaz az azonosítás sikeres volt, átlagolja az aktuális hangvektort az azonosítottal és HTTP 202 (Accepted) státuskóddal illetve a fejlécben az azonosított felhasználó nevével és egyedi azonosítójával válaszol.

- *authenticate()*: A */authenticate* végpontra küldött jelszó alapú autentikáció esetén hívódik meg. A klienstől érkező kérés tartalmazza az azonosítani kívánt felhasználó UID-t és a jelszót. Ellenőrzi, hogy a küldött jelszó megegyezik-e a korábban eltárolttal.

  Amennyiben igen, az autentikáció elején eltárolt hangvektort átlagolja a korábbival, majd HTTP 202 (Accepted) státuskóddal illetve a fejlécben az azonosított felhasználó nevével és egyedi azonosítójával válaszol.

  Ha a jelszavas autentikáció nem sikerült, törli az átlagolásra váró hangvektort és HTTP 403 (Forbidden) státuszkóddal válaszol a kliensnek.

- *model_loader.py*: A fejlesztés kezdetén a modell betöltése a *app.py*-ban volt implementálva. Mivel a Flask több szálat használ, problémát okozott, hogy a modellt valamelyik függvény hamarabb használta minthogy azt valóban betöltötte volna.

  Erre többféle megoldási lehetőség van. Például kötelezhetjük a Tensorflowt globális default gráf használatára. Mivel ez a megoldás csak részben működött, más javaslat alapján külön osztályba, a *ModelLoader*be szerveztem. Ez az osztály felelős a modell betöltéséért illetve ez végzi el a predikciót.

  - load(file_name): Betölti az paraméterként kapott útvonalon található modellt.
  - predict(model_input, model): A bemenet és a választott modell alapján visszaadja a prediktált kimenetet.

  A *ModelLoader* objektum inicializálásakor betölti a sziámi és a sima konvolúciós modellt is. Később a predict() függvényben paramétereiben állíthatjuk be, hogy melyik modellen szeretnénk a predikciót végrehajtani.

- *preprocess.py*:

  - *whiten(batch, rms)*: A *whiten* standardizálja a hangmintákat. Egy hangmintát egész számok sorozata reprezentál (PCM kódolás 4 vagy 16 biten). Először kivonja az átlagot minden számból, így a hangminta átlaga 0 lesz. Ezután újraskálázza a számokat, hogy ne legyenek kiugró amplitúdó értékek.

  - *preprocess_instances(downsampling, whitening)*: A *preprocess* függvénynek két fő paramétere van: a hangminta hossza és a *downsampling*, vagyis a csökkentett mintavételezési ráta. Ez a függvény végzi el a csökkentett mintavételezést és a

standardizálást egy hangmintán.

A csökkentett mintavételezés az eltérő mintavételi frekvencia miatt szükséges. Az Androidot futtató készülékek 8 és 16 bites PCM mellett 8, 16 és 44.1 kHz-es mintavételi frekvenciákat támogatnak. Esetünkben a modellt 4 kHz-en tanították a saját hangminták pedig 8 kHz-esek ezért 2-es csökkentett mintavételezés szükséges.

- *BatchPreProcessor(mode, instance_preprocessor, target_preprocessor)*: Ez az osztály wrapper funckiót lát el. Ez segít abban, hogy egységesen tudjuk kezelni az (inputs, outputs) klasszifikációs batcheket és a ([inputs_1, inputs_2], outputs) sziámi típusúakat is. A *mode* paraméterrel megadjuk, hogy milyen típusú batchek előfeldolgozását szeretnénk. Az *instance_preprocessor* a bemenetek, a *target_preprocessor* a kimeneteket preprocesszáló függvény. Például *BatchPreProcessor('classifier', preprocess_instances(downsampling))*.

- *preprocess(audio_file, sample_length, downsampling)*: Ez a függvény hatja végre a preprocesszálást. Beolvassa a paraméterként megadott hangmintát. A hangminta közepén *sample_length* hosszúságú darabot vág ki belőle. Létrehoz egy *BatchPreProcessor* objektumot, előfeldolgozza a batchet, majd visszatér a feldolgozott hangmintával.

  Mivel a jelenleg használt modell 3 másodperces és 4 kHz-en mintavételezett hangmintákat vár a bemenetére, a saját 4 másodperces hangmintákból 3 másodperces darabokat vág ki a közepétől számolva $1,5$-$1,5$ másodperces távolságra. Erre azért van szükség, hogy ha a felhasználó kicsit később kezd el beszélni vagy hamarabb hagyja abba, a felvétel elején vagy végén lévő szünet ne kerüljön bele a mintába.

5.3.3. Fejlesztési környezet

Ebben a fejezetben bemutatom a fejlesztéshez használt technológiai stacket. Szó lesz a szerver oldali és a kliens alkalmazáshoz használt toolokról, környezetekről.

5.3.3.1. Git

A Git egy nyílt forráskódú verziókezelő, ami a fejlesztésben rendkívül fontos. A verziókezelés nem csak akkor hasznos, ha többen dolgoznak egy projekten. Egyéni kódbázist refaktorálás után ezáltal visszaállíthatunk, törölt fájlokat, kódrészleteket követhetünk vissza. A saját repositorymat GitHubon tároltam.

5.3.3.2. Amazon EC2

A webszerver a 5.1 fejezet alapján Amazon EC2-es virtuális gépen fut. Az Amazon Elastic Compute Cloud szolgáltatás biztonságos és skálázható számítási kapacitást nyújt felhőben. Virtuális gépek bérelhetők rajta, amelyek utána testreszabhatók tárhely és erőforrások; processzor, memória szempontjából.

A felhasználók létrehozhatnak saját virtuális gépeket, amelyeken saját alkalmazásokat futtathatnak. Többféle fizetési modell közül választhatunk: Létezik óra vagy másodperc alapú, de saját dedikált gépet is bérelhetünk. Az AWS Educate segítségével

diákok korlátozottan de ingyenesen használhatják az Amazon EC2 szolgáltatásokat egy megadott kreditszintig.

A virtuális gépen Ubuntu 16.04 operációs rendszer fut és ssh-n keresztül férhetünk hozzá. Ehhez a gép létrehozásakor lehet új kulcspárt generálni vagy meglévőket használni. A gép bootolásakor a publikus kulcs bekerül a */.ssh/authorized_keys* fájlba. Később ssh-nál a saját privát kulcsunkat megadva tudunk autentikálin. A virtuális menedzselése az AWS Console-on keresztül történik. A gépen futó Flask szerver portján engedélyezni kellett a be és kimenő forgalmat. Ehhez létre kell hozni egy új *Security Groupot*, majd hozzáadni a megfelelő *Rule*-t.

Az alkalmazás szempontjából nagy előny, hogy a felhőbeli gépet könnyen elérjük, és fejlesztés közben sem a saját gépünk erőforrásait használja. Hátránya, hogy a kényelmes remote fejlesztés több konfigurációt igényel.

5.3.3.3. cmder

A cmder egy Windowshoz készült konzol emulátor. Beépített Gittel és ssh-klienssel rendelkezik és több Linuxos parancsot is támogat. Használata sokkal kényelmesebb a Puttynál, könnyedén csatlakozhatunk vele a Linuxos géphez.

Az ssh klienst paraméterezni kell a privát kulcsunkkal és a timeout elkerülése érdekében saját ssh konfigurációs fájllal. Eleinte probléma volt az ssh kapcsolat lejárási ideje, de ez kliens és szerveroldalon is könnyen konfigurálható. Linuxon a */etc/ssh/sshd_config*-ban a *ClientAliveInterval <seconds>* és *ClientAliveCountMax <seconds>*, Windowson (jelen esetben kliens oldalon) a paraméterben megadott konfigurációs fájlban *ServerAliveInterval <seconds>* és *ServerAliveCountMax <seconds>* beállításokkal küszöbölhető ki.

5.3.3.4. Anaconda

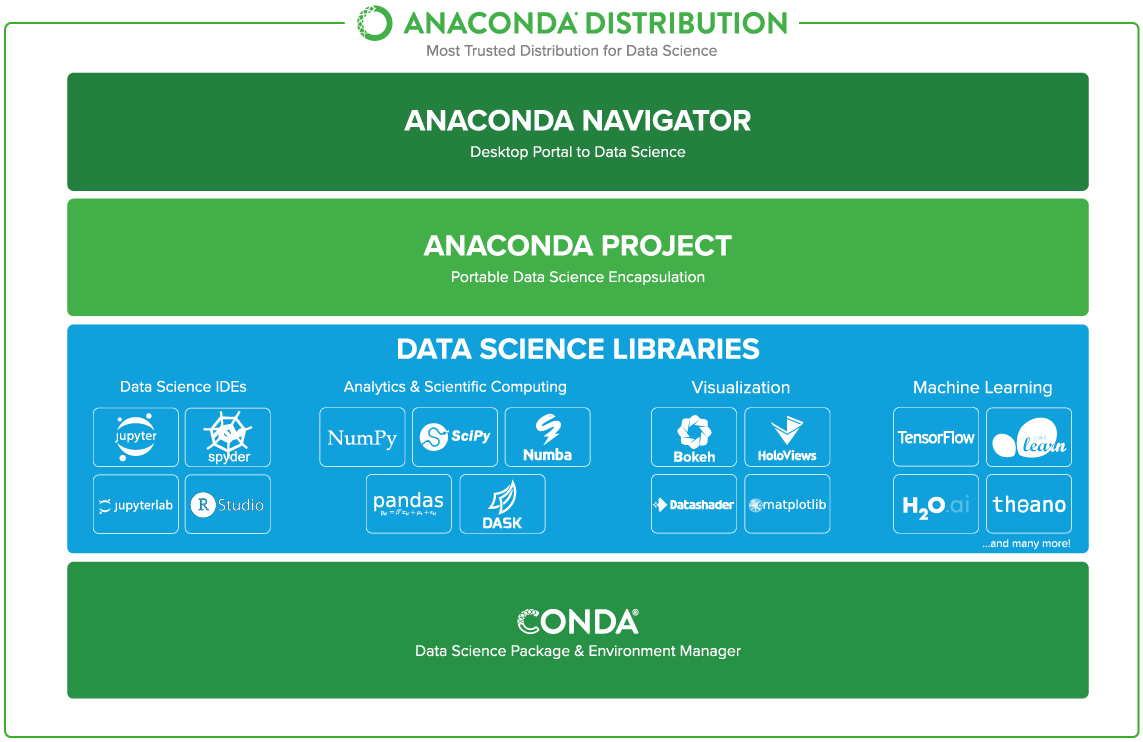
Az Anaconda egy nyílt forráskódú Python és R disztribúció, amely több mint 1500 programcsomaggal, saját csomagkezelővel, grafikus felülettel és virtuális környezetkezelővel rendelkezik.

Támogatja a gépi tanulást, tartalmazza a fontos gépi tanulási csomagokat (Keras, Tensorflow, Pytorch stb.), jupyter nootebookot. A virtuális környezetkezelővel a projekteknek külön virtuális környezetet hozhatunk létre. Ez fontos a verziók ütközésének elkerülése végett; Ha egy adott projektet Python 2.7-et használ, egy másik pedig Python 3.6-ot vagy ugyanolyan de eltérő verziójú programcsomagokat, akkor érdemes külön környezetet létrehozni nekik. A környezetek között a parancssorban navigálhatunk, csomagokat telepíthetünk rajtuk.

Az Anaconda Cloud további csomagokat, környezeteket, publikus és privát notebookokat tartalmaz. Ide feltölthetünk sajátot, illetve kereshetünk, telepíthetünk közülük csomagokat, környezeteket a saját gépünkre, környezetünkre.

5.3.3.5. JetBrains PyCharm

A JetBrain PyCharm egy Python programozási nyelvhez készült IDE. Azért esett erre a választás, mert van benne automatikus kódkiegészítés, intelligens kód navigáció - definícióra vagy használatra ugrás - és kód refaktorálás. Utóbbi a kód formázását és az



5.9. ábra. Flask szerver: Osztálydiagram.

importált csomagok rendezését jelenti.

Új projekt létrehozásakor saját virtuális környezetet hoz létre, de megadhatunk neki anaconda által létrehozottat, azt is képes kezelni. Jelen esetben a legfontosabb funkció a remote fejlesztés integrált támogatása volt.

A projekt beállításokban új Deployment létrehozásakor megadható a virtuális gép címe és a privát kulcs ssh-hoz. Megadhatók a lokális a remote mappák közötti mappingek és részletes szinkronizációs beállítások. A projekt interpreterének a remote gépen lévő Anaconda virtuális környezete által használt Python interpretert kell megadni, ez alapján a PyCharm feloldja a hozzá tartozó környezetet, látni fogja a programcsomagokat.

Szintén hasznos funckió a remote debugger, aminek segítségével a távoli virtuális gépen futó alkalmazást tudjuk debuggolni lokálisan.

5.3.3.6. Android Studio

A kliens oldali Android alkalmazás fejlesztéséhez Android Studiot használtam, ami az Android hivatalos fejlesztőkörnyezete. Az beépített templatek új *Activity*k és a Layout Editor nagyban megkönnyíti a felhasználói felület fejlesztését. Támogatja emulátorok használatát, amelyeken debugolni is lehet.

5.3.3.7. Genymotion

Mivel az Android Studio emulátorai nem támogatják teljes mértékben az AMD processzorokat, külső emulátort használtam, a *Genymotion*t. Ez egy külön alkalmazás és pluginnal csatlakoztatható az Android Studiohoz.

5.4. Tesztelés és kiértékelés

Az alkalmazást 8 emberrel teszteltem.

5.5. Használati esetek és továbbfejlesztési lehetőségek

Az implementált szoftver általános felhasználási módja a felhasználó azonosítás. Jelszó nélküli, csak hangalapú azonosítást kizárólag alacsony biztonsági szintű alkalmazásokban vagy funkciókban szabad használni. Minden komolyabb biztonságot követelő esetben vagy szigorú biztonsági küszöb, vagy kétfaktoros (hang és jelszó) autentikációt ajánlott használni.

-

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] Dean Nicolls. Digital identity verification blog. https://www.jumio.com/what-is-biometric-authentication/, Jul 2019.

[2] Gemalto. Biometrics: authentication and identification (definition, trends, use cases, laws and latest news). https://www.gemalto.com/govt/inspired/biometrics, 2019.

[3] David Van Leeuwen and Niko Brummer. An introduction to application-independent evaluation of speaker recognition systems. volume 4343, pages 330–353, 01 2007.

[4] John Hansen and Taufiq Hasan. Speaker recognition by machines and humans: A tutorial review. *Signal Processing Magazine, IEEE*, 32:74–99, 11 2015.

[5] Clark D. Shaver and John M. Acken. A brief review of speaker recognition technology. 2016.

[6] Finnian Kelly. *Automatic Recognition of Ageing Speakers*. PhD thesis, 04 2014.

[7] Nilu Singh, Alka Agrawal, and Prof. Raees Khan. Automatic speaker recognition: Current approaches and progress in last six decades. *Global Journal of Enterprise Information System*, 9:38–45, 07 2017.

[8] J. Garofolo, Lori Lamel, W. Fisher, Jonathan Fiscus, D. Pallett, N. Dahlgren, and V. Zue. Timit acoustic-phonetic continuous speech corpus. *Linguistic Data Consortium*, 11 1992.

[9] John Kominek and Alan W Black. Cmu arctic databases for speech synthesis. Technical report, Carnegie Mellon University, 2003.

[10] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: An asr corpus based on public domain audio books. pages 5206–5210, 04 2015.

[11] Arsha Nagrani, Joon Son Chung, and Andrew Zisserman. Voxceleb: A large-scale speaker identification dataset. 06 2017.

[12] Joon Son Chung, Arsha Nagrani, and Andrew Zisserman. Voxceleb2: Deep speaker recognition. pages 1086–1090, 09 2018.

[13] Chao Li, Xiaokong Ma, Bing Jiang, Xiangang Li, Xuewei Zhang, Xiao Liu, Ying Cao, Ajay Kannan, and Zhenyao Zhu. Deep speaker: an end-to-end neural speaker embedding system. 05 2017.

[14] Mirco Ravanelli and Y. Bengio. Speaker recognition from raw waveform with sincnet. pages 1021–1028, 12 2018.

[15] Weidi Xie, Arsha Nagrani, Joon Son Chung, and Andrew Zisserman. Utterance-level aggregation for speaker recognition in the wild. pages 5791–5795, 05 2019.

[16] Jonathan Boilard, Philippe Gournay, and R. Lefebvre. A literature review of wavenet: Theory, application and optimization. 03 2019.

[17] Oscar Knagg. voicemap. https://github.com/oscarknagg/voicemap, 2018.

[18] Oscar Knagg. Building a speaker identification system from scratch with deep learning. https://medium.com/analytics-vidhya/building-a-speaker-identification-system-from-scratch-with-deep-learning-f4c4a Oct 2018.

[19] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 815–823, June 2015.

[20] Olivier Moindrot. Triplet loss and online triplet mining in tensorflow. https://omoindrot.github.io/triplet-loss, 2018.

[21] Sycorax says Reinstate Monica (https://stats.stackexchange.com/users/22311/sycorax-says-reinstate monica). Why do neural networks need so many training examples to perform? Cross Validated. URL:https://stats.stackexchange.com/q/394128 (version: 2019-03-03).

[22] Jonathan Hui. Rl – meta-learning. https://medium.com/@jonathan_hui/meta-learning-how-we-address-the-shortcomings-of-our-deep-networks-a008aa4b5b2 Apr 2018.

[23] Lilian Weng. Meta-learning: Learning to learn fast. https://lilianweng.github.io/lil-log/2018/11/30/meta-learning.html, Nov 2018.

[24] Gregory R. Koch. Siamese neural networks for one-shot image recognition. 2015.

[25] Thomas Wolf. from zero to research — an introduction to meta-learning. https://medium.com/huggingface/from-zero-to-research-an-introduction-to-meta-learning-8e16e677f78a, Apr 2018.

[26] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. 03 2017.

[27] Alex Nichol and John Schulman. Reptile: a scalable metalearning algorithm. 2018.

[28] Matthijs Hollemans. Machine learning on mobile: on the device or in the cloud? https://machinethink.net/blog/machine-learning-device-or-cloud/, Febr 2017.