



---

**UNIVERSITATEA TEHNICĂ**  
DIN CLUJ-NAPOCA

---

**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE**  
**DEPARTAMENTUL CALCULATOARE**

## **RECUNOAȘTERE DE VORBITOR**

LUCRARE DE LICENȚĂ

Absolvent: **Maria-Alexandra MITIȘOR**

Coordonator științific: **Conf.dr.ing Anca MĂRGINEAN**

**2020**



**UNIVERSITATEA TEHNICĂ**  
DIN CLUJ-NAPOCA

**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE**  
**DEPARTAMENTUL CALCULATOARE**

DECAN,  
**Prof. dr. ing. Liviu MICLEA**

DIRECTOR DEPARTAMENT,  
**Prof. dr. ing. Rodica POTOLEA**

Absolvent: **Maria-Alexandra MITIȘOR**

**RECUNOAȘTERE DE VORBITOR**

1. **Enunțul temei:** Tema constă în dezvoltarea unui model capabil de a recunoaște persoana care vorbește dintr-o înregistrare dată.
2. **Conținutul lucrării:** Pagina de prezentare, Introducere, Obiectivele proiectului, Studiu bibliografic, Analiză și fundamentare teoretică, Proiectare de detaliu și implementare, Testare și validare, Manual de instalare și utilizare, Concluzii, Bibliografie
3. **Locul documentării:** Universitatea Tehnică din Cluj-Napoca, Departamentul Calculatoare
4. **Consultanți:** Conf.dr.ing Anca Mărginean
5. **Data emiterii temei:** 1 noiembrie 2019
6. **Data predării:** 8 iulie 2020

Absolvent: **Maria-Alexandra Mitîșor**

Coordonator științific: **Conf.dr.ing. Anca Mărginean**



**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE**  
**DEPARTAMENTUL CALCULATOARE**

**Declarație pe proprie răspundere privind  
autenticitatea lucrării de licență**

Subsemnata Maria-Alexandra Mitișor, legitimat(ă) cu CI seria CJ nr. 231294, CNP 2981001080033, autorul lucrării “Recunoaștere de vorbitor” elaborată în vederea susținerii examenului de finalizare a studiilor de licență la Facultatea de Automatică și Calculatoare, Specializarea Calculatoare și Tehnologia Informației din cadrul Universității Tehnice din Cluj-Napoca, sesiunea iulie a anului universitar 2019-2020, declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, pe baza cercetărilor mele și pe baza informațiilor obținute din surse care au fost citate, în textul lucrării, și în bibliografie.

Declar, că această lucrare nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române și a convențiilor internaționale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în fața unei alte comisii de examen de licență.

În cazul constatării ulterioare a unor declarații false, voi suporta sancțiunile administrative, respectiv, *anularea examenului de licență*.

Data  
8.07.2020

Nume, Prenume  
Mitișor Maria-Alexandra

Semnătura

---

## Cuprins

<b>Capitolul 1. Introducere – Contextul proiectului.....</b>	<b>6</b>
<b>Capitolul 2. Obiectivele Proiectului .....</b>	<b>8</b>
2.1 Metricile din machine learning .....	8
<b>Capitolul 3. Studiu Bibliografic .....</b>	<b>11</b>
<b>Capitolul 4. Analiză și Fundamentare Teoretică.....</b>	<b>13</b>
4.1 Moduri în care putem trata sunetul .....	13
4.1.1 Time series (serie temporală).....	14
4.1.2 Spectrograme (mfcc sau melspectrogramă).....	14
4.2 Elemente de machine learning .....	17
4.2.1 Model; Antrenament; testare.....	17
4.2.2 Clasificare .....	17
4.2.3 Loss .....	17
4.2.4 Learning rate .....	17
4.2.5 Class weights .....	18
4.2.6 Neuron, rețea neuronală .....	18
4.2.7 Regularizare .....	18
4.2.8 Funcție de activare .....	19
4.3 Tipuri de rețele neuronale .....	19
4.3.1 RNN .....	20
4.3.2 CNN .....	22
4.3.3 Transfer Learning.....	25
4.3.4 MobileNet v2 .....	25
4.3.5 DenseNet.....	26
4.3.6 Comparație între MobileNet și DenseNet.....	27
<b>Capitolul 5. Proiectare de Detaliu si Implementare.....</b>	<b>29</b>
5.1 Extragerea de features din clipurile audio .....	29
5.2 Împărțirea dataframe-ului în setul de date de train și setul de date de test .	31
5.3 Construirea modelului.....	32
5.4 Compilarea modelului.....	35
5.5 Antrenarea modelului.....	35
5.6 Evaluarea modelului folosind metricile explicate în capitolul 2 .....	36
5.6.1 display_metrics .....	36

---

5.6.2 evaluate_model .....	37
5.6.3 plot_history .....	38
5.6.4 plot_compare_val_loss .....	39
5.7 Furnizarea către model a unor clipuri străine și detectarea vorbitorilor din acestea.....	40
<b>Capitolul 6. Testare și Validare.....</b>	<b>42</b>
6.1 CNN1 .....	42
6.2 CNN2 .....	43
6.3 CNN3 .....	44
6.4 CNN4 .....	45
6.5 CNNCw.....	46
6.6 CNNCwHTK .....	47
6.7 CNNCwSr .....	48
6.8 CNNMF .....	49
6.9 CNNMFDiff.....	50
6.10 CNN2Men.....	50
6.11 CNNTwoInputs.....	51
6.12 MobileNet1 .....	52
6.13 MobileNet2 .....	53
6.14 DenseNet1 .....	54
6.15 DenseNetMelSpec.....	54
6.16 DenseNet2Men .....	56
6.17 DenseNetMF .....	56
6.18 DenseNetMFDiff .....	57
6.19 Verificarea pe date noi .....	57
<b>Capitolul 7. Manual de instalare și utilizare.....</b>	<b>59</b>
<b>Capitolul 8. Concluzii.....</b>	<b>61</b>
<b>Bibliografie.....</b>	<b>62</b>

## Capitolul 1. Introducere – Contextul proiectului

Acest proiect a fost elaborat în ideea în care identificarea vorbitorilor dintr-o înregistrare ar putea fi utilă poliției, eventual pentru a descoperi cine a săvârșit o anumită infracțiune (prin analiza înregistrării unor apeluri telefonice sau a unor înregistrări cu vocea suspectului).

O altă posibilă utilizare a unui sistem de recunoaștere de vorbitor precum cel din proiect ar fi securizarea anumitor dispozitive sau incinte, prin acordarea permisiunii de utilizare a dispozitivului / de intrare în incintă numai a persoanei a cărei voce se potrivește cu cea din baza de date a sistemului. Acest mecanism este similar celui de recunoaștere facială implementat deja pe unele telefoane inteligente și tablete. Mecanismul se numește verificare de vorbitor și înseamnă că sistemul preia o mostră de vorbire de la o persoană și returnează un verdict de tipul “Este persoana X” sau “Nu este persoana X”. Pentru această lucrare am dezvoltat un sistem de recunoaștere de vorbitor (adică un sistem care să spună cine vorbește în înregistrare, alegând din mai mulți posibili vorbitori), și nu unul de verificare de vorbitor.

Acest sistem funcționează independent de ceea ce spune vorbitorul (astfel de sisteme se numesc text-independent). Există și sisteme care pot recunoaște o persoană doar dacă aceasta spune anumite cuvinte-cheie, acestea sunt sisteme text-dependent.

Abordarea aleasă pentru atingerea scopului proiectului a fost una bazată pe machine learning, mai exact prin utilizarea rețelelor neuronale convoluționale (CNN) sau a rețelelor neuronale pre-antrenate (transfer learning). Am pornit de la o rețea convoluțională simplă, am modificat progresiv diverși parametri pentru a obține rezultate cât mai bune, iar apoi am făcut experimente cu două tipuri de rețele pre-antrenate, DenseNet și MobileNet.

Pentru antrenamentul rețelelor am utilizat setul de date VoxCeleb, care conține înregistrări aparținând a peste 1000 de celebrități, primele 20 fiind folosite în proiect. Conform sursei [12], acest set de date a fost obținut prin tehnici de recunoaștere facială, preluând filmulețe de pe youtube în care apar vorbitorii recunoscuți și tăind în mod automat clipuri audio scurte în care aceștia vorbesc. Modul în care au fost extrase clipurile pe baza recunoașterii faciale este ilustrat în figura 1.1, preluată din sursa [12]:

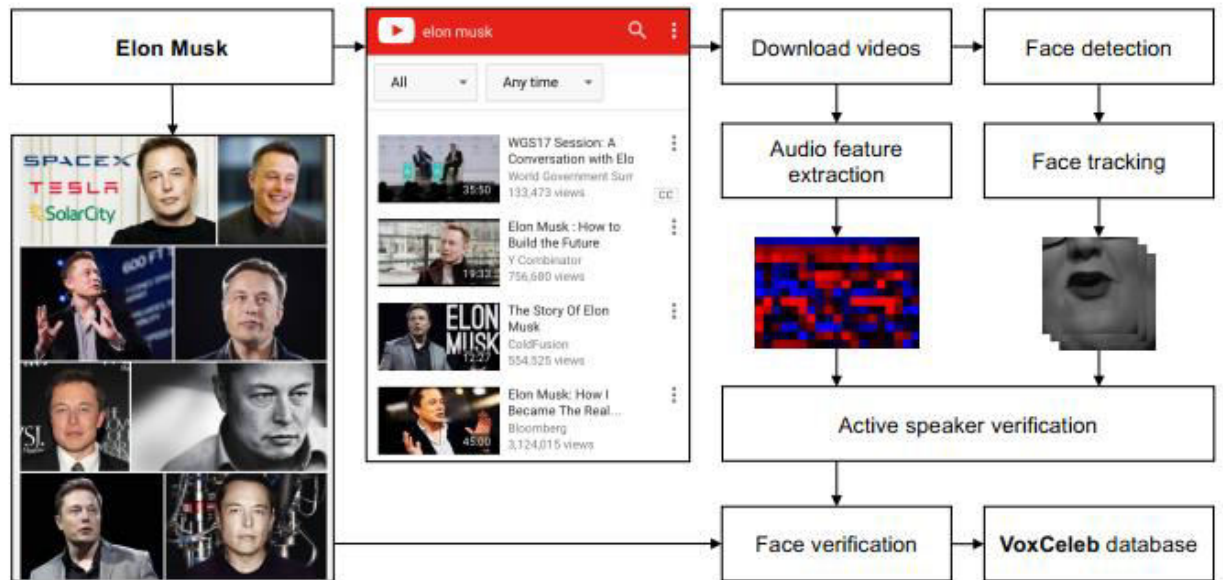


Figura 1. 1: Extragerea clipurilor audio din VoxCeleb

În cazul în care proiectul ar fi folosit în realitate, aplicației ar trebui să i se furnizeze un set de date de antrenament (înregistrări cu posibili suspecți, cât mai multe înregistrări de la cât mai multe persoane, care să conțină cât mai puțin zgomot), pentru a putea învăța să asocieze caracteristicile unei voci cu o anumită persoană. Astfel, rețeaua neuronală va clasifica o înregistrare care va veni din exterior într-una din clasele învățate până acum, calculând probabilitățile ca noua înregistrare să facă parte din fiecare clasă, iar clasa căreia îi va aparține în final va fi cea cu probabilitatea cea mai mare. Așadar, se va putea identifica persoana care vorbește (suspectul, eventual), cu un anumit grad de acuratețe.

Pentru recunoașterea vocii unei anumite persoane este nevoie să se identifice niște proprietăți ale vocii acesteia. Acest proiect a utilizat pentru extragerea proprietăților melspectrogramele și mfcc-urile, concepte care vor fi detaliate într-unul din capitolele următoare. Aceste melspectrograme/mfcc-uri sunt niște imagini, deci folosindu-le am transformat problema clasificării sunetelor într-una de clasificare de imagini.

## Capitolul 2. Obiectivele Proiectului

Obiectivul principal al acestui proiect este să recunoască cât mai multe dintre persoanele care vorbesc în înregistrările furnizate. Datele de intrare furnizate rețelelor neuronale au fost niște imagini extrase din clipurile audio pe baza unor transformări descrise într-un capitol ulterior. Aceste imagini poartă numele de mfcc-uri, respectiv melspectrograme. Corectitudinea și performanța rețelelor neuronale implementate a fost măsurată folosind metricile uzuale din machine learning, explicate în continuare.

### 2.1 Metricile din machine learning

Metricile folosite pentru a evalua acest proiect sunt acuratețea, recall-ul, precizia și matricea de confuzie (confusion matrix). Pentru calculul metricilor, e nevoie de înțelegerea anumitor termeni, și anume TP (true positive, numărul de mostre care aparțin clasei X și au fost identificate corect ca aparținând clasei X), FP (false positive, numărul de mostre din alte clase identificate ca fiind în clasa X), TN (numărul de mostre din alte clase care au fost identificate corect ca nefăcând parte din clasa X) și FN (numărul de mostre care fac parte din clasa X, dar nu au fost identificate ca făcând parte din clasa X). În continuare, voi detalia semnificația fiecărei metrici amintite:

acuratețea (accuracy) =  $(TP + TN) / (TP + TN + FP + FN)$  : numărul de mostre (exemple) clasificate corect supra numărul total de mostre clasificate

recall =  $TP / (TP + FN)$  : numărul de mostre corect clasificate ca făcând parte din clasa X supra numărul total de mostre care chiar fac parte din clasa X (va fi mai mic dacă ratăm multe exemple care făceau, de fapt, parte din clasa X)

precizia (precision) =  $TP / (TP + FP)$  : numărul de mostre corect clasificate ca făcând parte din clasa X supra numărul total de mostre clasificate ca fiind din clasa X (va fi mai mic dacă avem multe exemple greșit clasificate ca fiind X).

matricea de confuzie (confusion matrix) : un tabel care pe verticală și pe orizontală are toate clasele, iar în celula (X, Y) are numărul de exemple din clasa X care au fost clasificate în clasa Y). Un exemplu avem în figura 2.1, după cum se vede, o rețea antrenată bine are valorile cele mai mari din matricea de confuzie pe diagonala secundară:



Target	Selected									Acc
	1	2	3	4	5	6	7	8	9	
1	<b>137</b>	13	3	0	0	1	1	0	0	0.89
2	1	<b>55</b>	1	0	0	0	0	6	1	0.86
3	2	4	<b>84</b>	0	0	0	1	1	2	0.89
4	3	0	1	<b>153</b>	5	2	1	1	1	0.92
5	0	0	3	0	<b>44</b>	2	2	1	2	0.82
6	0	0	2	1	4	<b>35</b>	0	0	1	0.81
7	0	0	0	0	0	0	<b>61</b>	2	2	0.94
8	0	0	0	1	0	0	0	<b>69</b>	3	0.95
9	0	0	0	0	0	0	0	2	<b>26</b>	0.93
										<b>0.89</b>

The diagonal elements (correct decisions) are marked in bold. Column "Acc" provides the specific accuracy for each key.

Figura 2. 1: Confusion matrix; sursa: [www.google.com](http://www.google.com)

Așadar, obiectivul inițial al proiectului a fost îmbunătățirea acestor metrici (obținerea unor valori cât mai apropiate de 1), pentru o clasificare cât mai exactă. După atingerea acestui obiectiv, atingerea țelului principal a fost verificată și prin furnizarea de clipuri străine modelului (clipuri care nu au fost nici în setul de date de antrenament, nici în setul de date de validare) și urmărirea rezultatului pe care acesta îl dă.

Acest obiectiv principal al proiectului a fost împărțit în două obiective secundare, mai specifice, și anume găsirea unei reprezentări a datelor optimă, care să faciliteze învățarea, și găsirea unei arhitecturi de rețea care să clasifice cât mai corect datele de intrare. Pentru aceasta, am ales ca și intrări mfcc-urile și melspectrogramele (descrise într-un capitol care urmează), iar ca și arhitecturi de rețea am încercat trei tipuri de arhitecturi: rețea neuronală convoluțională simplă, rețea neuronală bazată pe rețeaua pre-antrenată MobileNet și rețea bazată în mod similar pe DenseNet.

De asemenea, pentru a atinge obiectivele descrise mai sus, am modificat pe rând anumiți parametri ai imaginilor (dimensiuni, nivel de detaliu) și ai rețelelor sau ai procesului de învățare (learning rate, batch size, funcție de optimizare, număr de epoci pentru antrenament).

În extragerea mfcc-urilor am căutat ca dimensiunile imaginilor extrase să corespundă lungimii clipurilor (să nu se introducă pe lângă informația relevantă mulți biți de umplură/padding) și ca informația să fie eșantionată la o rată suficient de bună pentru a cuprinde caracteristicile cele mai importante ale timbrului vorbitorului.

Am ales arhitecturile rețelelor neuronale luând în considerare proiecte similare realizate de alții, cum ar fi cel din sursa [4]. Pe lângă acest aspect, am studiat arhitecturi uzuale de astfel de modele de rețele neuronale și am înțeles principiile pe care acestea se bazează. Astfel, am reușit în mod progresiv să mă apropiu de obiectivele setate: cu fiecare experiment realizat, am schimbat câte un parametru al modelului și am observat rezultatele obținute, apoi am tras anumite concluzii legate de efectul unor parametri și de ceea ce trebuie schimbat în continuare.

La final, pentru a verifica faptul că obiectivul proiectului menționat inițial a fost atins, am testat funcționalitatea acestuia pe date noi. Am selectat diverse clipuri cu interviuri ale primelor 10 celebrități din setul de date, având grijă ca acestea să nu se suprapună cu cele din setul de antrenament (am preluat pe cât posibil interviuri de după 2017, anul în care a apărut setul de date VoxCeleb). Rezultatele nu au fost chiar la fel de bune ca și pe setul de test, deoarece clipurile din setul de test sunt preluate din aceleași interviuri ca și cele din setul de antrenament și, deci, calitățile vocii sunt similare.

### Capitolul 3. Studiu Bibliografic

Până acum au fost create mai multe sisteme de recunoaștere de vorbitor bazate pe setul de date VoxCeleb, cu diverse rezultate. În acest capitol, voi prezenta 3 dintre aceste sisteme, împreună cu rezultatele la care au ajuns fiecare.

Prima soluție este cea oferită de autorii articolului [15], cu o acuratețe maximă de 58%. Autorii au antrenat modelul lor pe 8 vorbitori și au folosit numai o parte din clipurile din setul de date (70%). Metoda lor se bazează tot pe mfcc-uri, și pe Deep Neural Networks sau SVM (Support Vector Machines). Modelul are 3 layere cu funcția de activare ReLU, un layer de dropout cu valoarea de 0.5 și un layer de clasificare. Acuratețea de 58% a fost obținută prin metoda SVM, metoda DNN obținând numai 43% acuratețe.

În articolul [16], autorii descriu un alt sistem de recunoaștere de vorbitor. Aceștia au realizat mai multe experimente bazate pe rețele neuronale convoluționale deep cum ar fi VGG și ResNet, peste care au adăugat un layer de self-attention (un tip de layer util în clasificare, descris în detaliu de către ei), un layer de average pooling și unul de clasificare cu softmax. Arhitectura rețelei propusă de către autori este cea din figura 3.1.

Rețeaua lor a fost antrenată pe 1251 de vorbitori (toți), și metoda folosită pentru extragerea de features este tot mfcc. În aceste condiții, acuratețea top-1 obținută a fost de 90.8%, iar top-5 de 96.5% (acuratețea top-n înseamnă că un exemplu se presupune a fi clasificat corect dacă probabilitatea corespunzătoare clasei corecte se află în topul primelor n probabilități rezultate din model pentru toate clasele).

Al treilea model de rețea antrenată pe setul de date VoxCeleb este cel din sursa [12]. Și această rețea a fost antrenată pe 1251 de vorbitori, iar arhitectura este bazată pe convoluție (layere convoluționale). Ca și imagini de intrare, autorii au extras spectrograme cu un canal, din 3 secunde de vorbire, cu dimensiuni de 512x300.

Arhitectura rețelei neuronale este bazată pe CNN-ul VGG și este prezentată mai concret în figura 3.2.

Rezultatele obținute sunt de 80.5% acuratețe top-1 și 92.1% acuratețe top-5.

Layer	Support	Filt dim.	# filts.	Stride	Data size
conv1	7×7	1	96	2×2	254×148
mpool1	3×3	-	-	2×2	126×73
conv2	5×5	96	256	2×2	62×36
mpool2	3×3	-	-	2×2	30×17
conv3	3×3	256	384	1×1	30×17
conv4	3×3	384	256	1×1	30×17
conv5	3×3	256	256	1×1	30×17
mpool5	5×3	-	-	3×2	9×8
fc6	9×1	256	4096	1×1	1×8
apool6	1×n	-	-	1×1	1×1
fc7	1×1	4096	1024	1×1	1×1
fc8	1×1	1024	1251	1×1	1×1

Figura 3. 1: Arhitectura rețelei propusă în sursa [12]

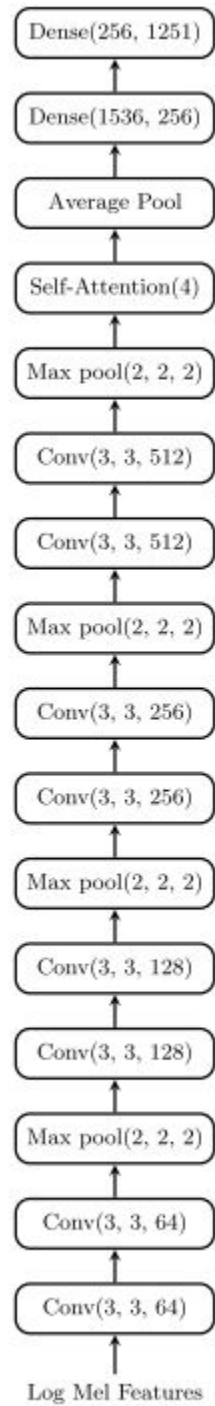


Figura 3. 2: Arhitectura rețelei propusă în sursa [15]

Merită menționat faptul că autorii articolului [12] au folosit tehnica de augmentare a datelor, ceea ce presupune introducerea în setul de antrenament a unor exemple obținute artificial din cele originale (de exemplu, prin oglindirea unei imagini) pentru a descuraja învățarea de către rețea a unor caracteristici care aparțin doar setului de antrenament și nu clasei în general.

## Capitolul 4. Analiză și Fundamentare Teoretică

### 4.1 Moduri în care putem trata sunetul

Conform [11], sunetul este o undă longitudinală (sau o mulțime de unde longitudinale) care constă în compresii și rarefieri ale mediului pe care îl traversează. Sunetul poate fi caracterizat în mai multe moduri, printre care amintim: lungimea de undă, amplitudinea, perioada și frecvența.

**Lungimea de undă (ilustrată în figura 4.1.1):**

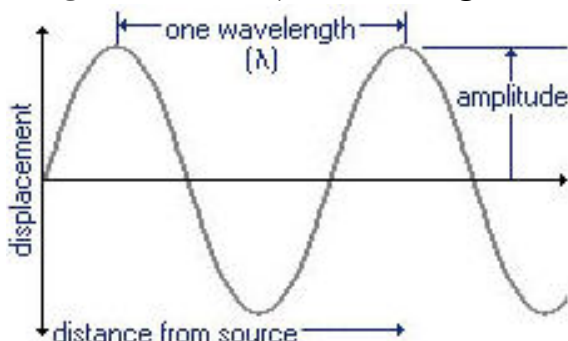


Figura 4. 1.1: Lungimea de undă; sursa: [11]

Distanța minimă la care unda se repetă se numește “lungime de undă” și se notează prin litera grecească lambda ( $\lambda$ ). Vârfurile părților concave ale unde sunt compresiile, iar vârfurile părților convexe sunt rarefierile. Astfel, distanța dintre două compresii consecutive sau două rarefieri consecutive este lungimea de undă. Un ciclu complet constă într-o compresie și o rarefiere, consecutive.

**Amplitudinea:**

Atunci când o undă trece printr-un mediu, particulele mediului sunt dislocate temporar față de poziția lor inițială. Valoarea maximă a acestei dislocări este amplitudinea sunetului. Amplitudinea poate fi văzută și ca mărimea (înălțimea față de axa orizontală) unde.

**Perioada:**

Timpul necesar producerii unui ciclu complet al unde se numește perioadă, și este notată cu  $T$ .

**Frecvența (ilustrată în figura 4.1.2):**

Numărul de cicluri complete produse într-o unitate de timp (secundă) se numește frecvența sunetului. Unitatea de măsură a frecvenței este Hertz (Hz).  $1\text{ Hz} = \text{un ciclu pe secundă}$ .

Relația dintre frecvență și perioadă este următoarea:  $f = 1/T$ .

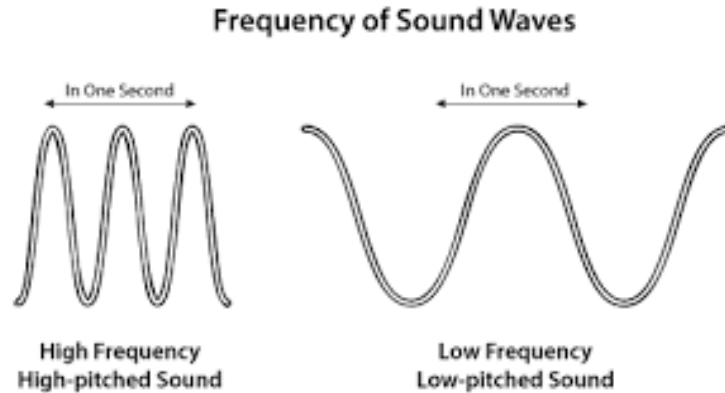


Figura 4. 1.2: Frecvența sunetului; sursa: [11]

#### 4.1.1 Time series (serie temporală)

O serie temporală este un șir de valori ale unei mărimi (temperatură, presiune, sau amplitudinea sunetului, de exemplu) aranjate după timp. După cum am sugerat și mai devreme, sunetul poate fi văzut ca o serie temporală dacă considerăm amplitudinea (valoarea semnalului) ca fiind valoarea indexată în funcție de timp.

#### 4.1.2 Spectrograme (mfcc sau melspectrogramă)

În realitate, sunetul este compus din mai multe unde cu frecvențe diferite. Spectrogramele sunt un mod de a reprezenta sunetul printr-o imagine, aplicându-i acestuia transformata Fourier (care va transforma unda din domeniul timp în domeniul frecvență). Melspectrogramele sunt asemănătoare cu spectrogramele, doar că acestea folosesc scara Mel pentru reprezentarea frecvenței (acest lucru va fi detaliat în continuare. În cele ce urmează, voi detalia modul de obținere al spectrogramelor, conform [2].

Conform informațiilor despre sunet prezentate mai devreme, unda este reprezentată în domeniul timp, deci axa verticală reprezintă amplitudinea, iar cea orizontală reprezintă timpul. Putem împărți unda în mai multe bucăți scurte, care vor fi procesate separat, iar la final rezultatele vor fi unite (concatenate), astfel obținându-se spectrograma undei.

Prima etapă de procesare constă în aplicarea transformatei Fourier (FFT în figura 4.1.3) pe fiecare sector de undă, astfel trecând în domeniul frecvență. Obținem câte un spectru de frecvențe pentru fiecare sector de undă:

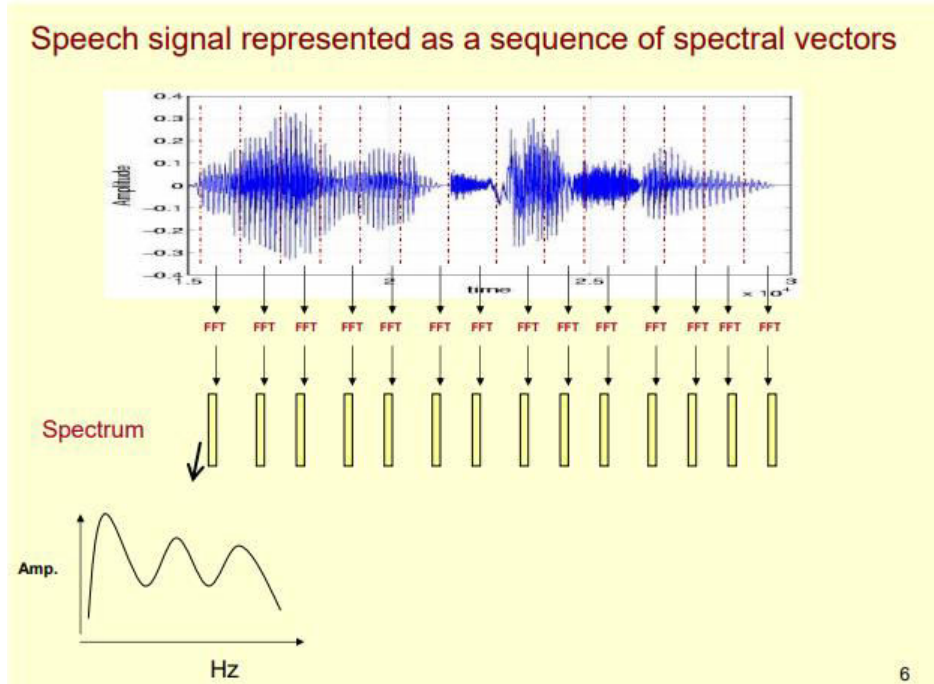


Figura 4.1. 3: Prima etapă; sursa: [2]

Apoi, rotim cu 90 de grade spre stânga graficele (bucățile de undă în domeniul frecvență) obținute și mapăm valorile amplitudinii la niveluri de gri între 0 și 255 (vezi figura 4.1.4):

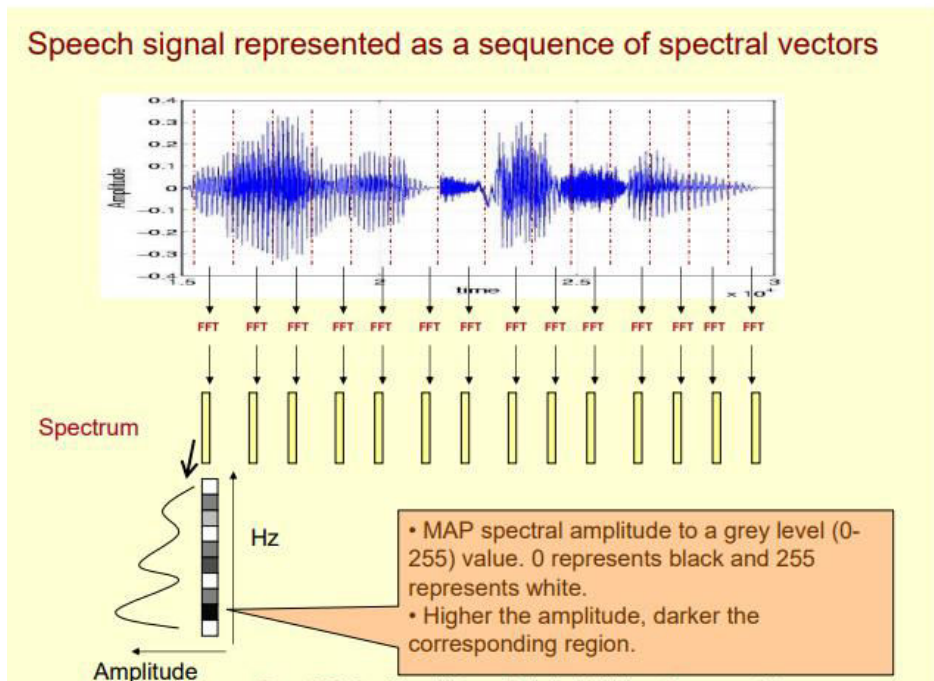


Figura 4. 1.4: A doua etapă; sursa: [2]

În final, concatenăm vectorii cu valori între 0-255 obținuți, astfel compunând spectrograma (figura 4.1.5):

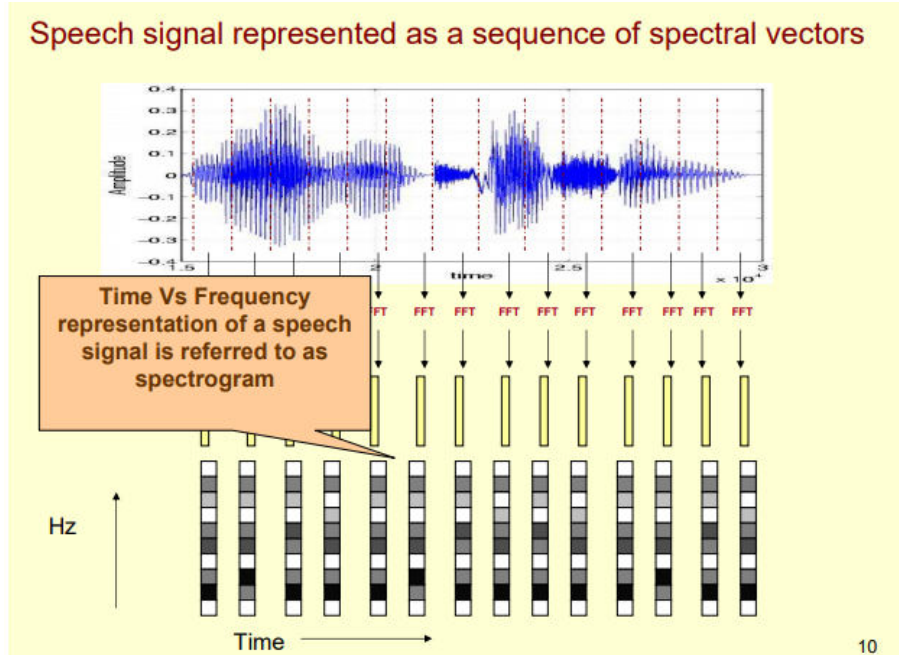


Figura 4. 1.5: Spectrograma; sursa: [2]

Pentru a calcula melspectrogramele, frecvența trebuie măsurată pe scara Mel, și nu în Hz. Conform Wikipedia, această scară a apărut din cauza unei observații făcute în 1937 de către Stevens, Volkman și Newman, care constă în faptul că urechea umană percepe diferențele de frecvență (faptul că un sunet este mai înalt decât altul) mult mai ușor într-o anumită zonă de frecvențe. Mai exact, două sunete foarte înalte (aproximativ peste 500 Hz), care sunt la o distanță de o octavă unul față de celălalt, sunt percepute ca fiind mult mai apropiate, pe când două sunete din zona confortabilă a urechii umane la distanță de o octavă sunt clar percepute ca fiind diferite. După pragul de 500 Hz, patru octave pe scara Hz sunt percepute ca fiind doar două octave (pe scara Mel). Numele Mel vine de la cuvântul melody, iar una din cele mai populare formule prin care se trece de la scara Hz la scara Mel este cea din figura 4.1.6:

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

Figura 4. 1.6: Formula de trecere la scara Mel; sursa: Wikipedia

Așadar, melspectrograma reprezintă o spectrogramă cu frecvențele transpuse în scara Mel.

În final, modul cel mai întâlnit prin care se trece de la sunet la imagine este mfcc (Mel Frequency Cepstral Coefficients). Alături de melspectrogramă, aceasta este metoda pe care am folosit-o în acest proiect pentru a extrage caracteristicile sunetului. Diferența dintre cele două constă în faptul că mfcc este o melspectrogramă pe care aplicăm transformata cosinus discretă (DCT).



## 4.2 Elemente de machine learning

Urmează să detaliez câțiva termeni din machine learning folosiți în continuarea acestei lucrări.

### 4.2.1 Model; Antrenament; testare

Un model reprezintă o entitate capabilă de a utiliza niște informații pe care le-a învățat. Învățarea se produce în procesul de antrenament, pe un anumit set de date care trebuie să fie disjunct cu cel folosit pentru testarea și validarea modelului. La antrenament se furnizează modelului exemple, alături de clasa din care fac parte, pentru ca acesta să învețe apoi să clasifice exemple noi. La testare/validare se măsoară cât de bine a învățat modelul să clasifice exemple care nu se regăsesc în setul de antrenament.

### 4.2.2 Clasificare

Alături de regresie, care presupune prezicerea cât mai corectă unei valori, clasificarea este unul dintre scopurile principale ale machine learning-ului. Aceasta presupune încadrarea unui exemplu nou de către model într-una din clasele pe care a fost antrenat.

### 4.2.3 Loss

Loss-ul este, în esență, diferența dintre valoarea de ieșire prezisă și cea corectă.

### 4.2.4 Learning rate

Learning rate-ul este viteza (pasul) cu care înaintăm pe curba din figura 4.2.1 (preluată din sursa [14]) pentru a atinge valoarea cea mai mică pentru loss. Dacă această valoare este prea mică, modelul se va antrena foarte încet, dar dacă este prea mare este posibil să trecem peste punctul optim de pe curbă fără să îl descoperim.

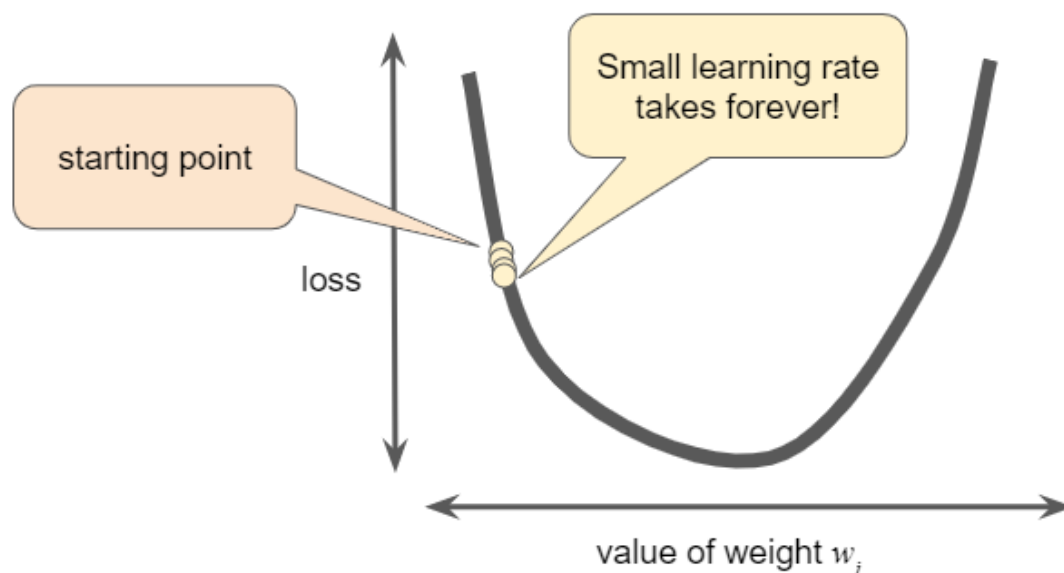


Figura 4.2. 1: Learning rate

### 4.2.5 Class weights

Din cauza faptului că unele seturi de date nu sunt echilibrate (conțin semnificativ mai multe exemple pentru unele clase decât pentru altele), clasificarea poate suferi. Dacă folosim class weights, se atribuie o pondere (deci o importanță) mai mare exemplorilor mai puține, astfel echilibrând modelul.

### 4.2.6 Neuron, rețea neuronală

Un neuron reprezintă o celulă de procesare a informației care are una sau mai multe intrări (inputs) și una sau mai multe ieșiri (outputs). O rețea neuronală este formată din mai multe layere (straturi) de neuroni, unde intrările unor neuroni sunt ieșirile altora (astfel, ei sunt conectați).

Rețelele neuronale pot fi reprezentate sub forma unor funcții ca și cea din figura 4.2.2, unde coeficienții cu care sunt înmulțite valorile de ieșire ale neuronilor ( $w_i$ ) se numesc “weights”.

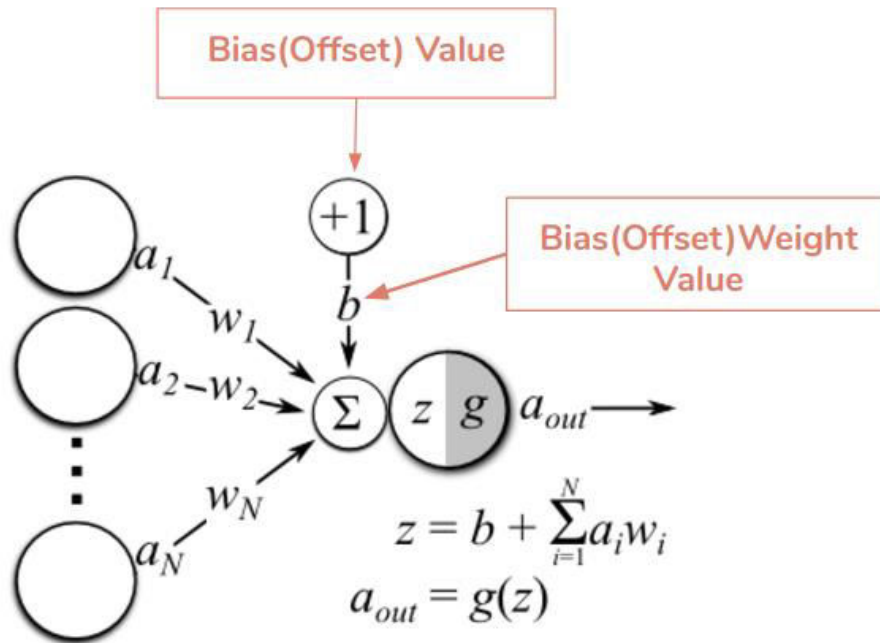


Figura 4.2. 2: Rețea neuronală

### 4.2.7 Regularizare

Pentru a evita crearea de modele complexe care să învețe caracteristici ale datelor valabile doar pe setul de antrenament (fenomen numit overfitting), se folosește regularizarea. Acest procedeu penalizează modelele care au termenul de regularizare prea mare. Două tipuri populare de regularizare sunt L1 (termenul de regularizare e suma modulelor weight-urilor) și L2 (termenul de regularizare e suma pătratelor weight-urilor).

### 4.2.8 Funcție de activare

Funcția de activare reprezintă transformarea care se aplică în interiorul unui neuron intrării, astfel rezultând ieșirea. O funcție de activare, deci, are forma  $\text{output} = F(\text{input})$ . Un exemplu de funcție de activare des întâlnită este ReLU (rectified linear unit), unde ieșirea este egală cu intrarea, dacă aceasta e pozitivă, și e egală cu zero altfel. Graficul acestei funcții este reprezentat în figura 4.2.3:

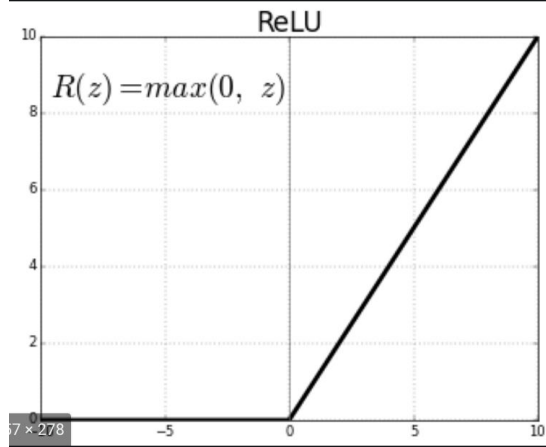


Figura 4.2. 3: ReLU

O altă funcție de activare utilizată în acest proiect este softmax, și se folosește pentru clasificare. Modul de utilizare al acestei funcții este ilustrat în figura 4.2.4, preluată din sursa [14]:

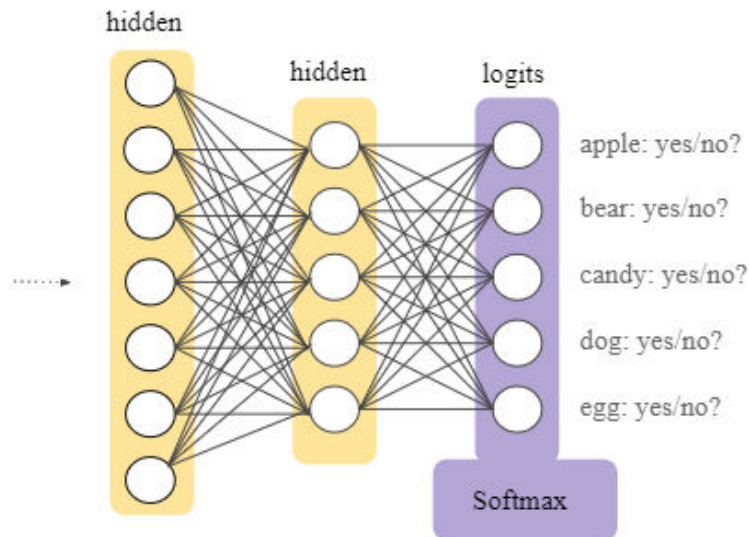


Figura 4.2. 4: Softmax

### 4.3 Tipuri de rețele neuronale

În continuare voi prezenta două tipuri de rețele neuronale folosite pentru a clasifica sunete: rețele neuronale recurente (RNN, Recurrent Neural Network) și rețele neuronale convoluționale (CNN, Convolutional Neural Network). De asemenea, va fi explicat conceptul de transfer learning și vor fi prezentate și comparate două arhitecturi

ale unor rețele pre-antrenate, folosite în experimentele din cadrul acestui proiect, și anume DenseNet și MobileNet.

### 4.3.1 RNN

Acest tip de rețele este folosit în general împreună cu o reprezentare a datelor sub forma unei serii temporale. Sursa [10] descrie neuronul recurent ca fiind un neuron care la momentul de timp  $t_i$  primește ca și intrare ieșirea lui de la momentul  $t_{i-1}$ . Pentru a reprezenta acest lucru, se aplica procedeul de “desfășurare în timp”, ilustrat în figura 4.3.1:

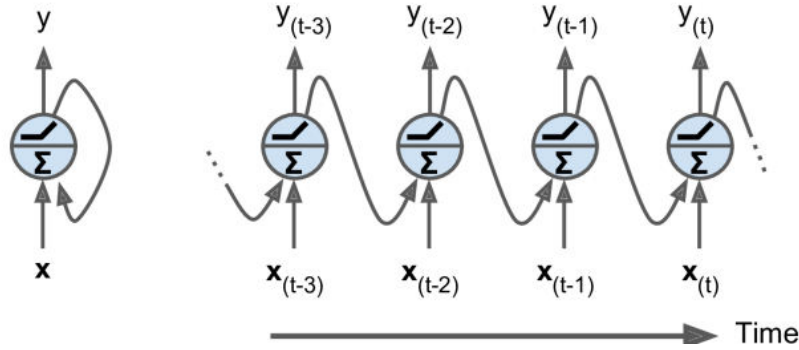


Figura 4.3. 4: Desfășurarea în timp; sursa: [10]

Pentru a crea un layer cu astfel de neuroni, trebuie ca intrările fiecărui neuron să fie ieșirea neuronului anterior și ieșirea neuronului curent cu un moment de timp în urmă, ca și în figura 4.3.2:

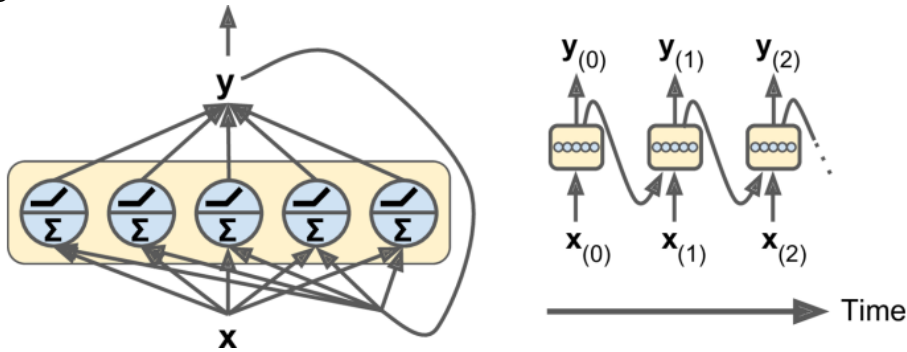


Figura 4.3. 5: Layer cu neuroni recurenți; sursa: [10]

Formula cu care se calculează ieșirea unui neuron recurent este cea din figura 4.3.3, iar, similar, cu formula din figura 4.3.4 se calculează vectorul de ieșire pentru un layer format din mai mulți neuroni recurenți:

$$\mathbf{y}(t) = \phi\left(\mathbf{x}(t)^T \cdot \mathbf{w}_x + \mathbf{y}(t-1)^T \cdot \mathbf{w}_y + b\right)$$

Figura 4.3. 6: Calculul ieșirii pentru un singur neuron; sursa:[10]

În formula următoare  $X$ ,  $Y$ ,  $W_x$  și  $W_y$  sunt matrice de dimensiune  $m \times n_i$ ,  $m \times n_n$ ,  $n_i \times n_n$ , respective  $n_n \times n_n$ , unde  $m$  este numărul de instanțe din batch,  $n_n$  este numărul de neuroni din layer, iar  $n_i$  este numărul de input features.

$$Y_{(t)} = \phi(X_{(t)} \cdot W_x + Y_{(t-1)} \cdot W_y + b)$$

$$= \phi\left(\begin{bmatrix} X_{(t)} & Y_{(t-1)} \end{bmatrix} \cdot W + b\right) \text{ with } W = \begin{bmatrix} W_x \\ W_y \end{bmatrix}$$

Figura 4.3. 7: Calculul ieșirii pentru mai mulți neuroni; sursa:[10]

Acest tip de rețele este, în cele mai multe cazuri, folosit pentru regresie (prezicerea următoarei valori dintr-o serie temporală), dar o RNN poate fi folosită și în clasificare dacă layerelor recurente li se adaugă un layer dens/fully connected (toți neuronii sunt conectați între ei) cu funcția de activare softmax. Această funcție normalizează vectorul de ieșire obținut de la layerul anterior și îl transformă într-o distribuție de probabilitate. Fiecărui neuron din acest layer cu softmax îi corespunde o clasă, astfel că clasa corespunzătoare neuronului cu ieșirea (probabilitatea) cea mai mare va fi cea în care vom clasifica intrarea rețelei. O RNN cu layer dens și softmax este prezentată în figura 4.3.5:

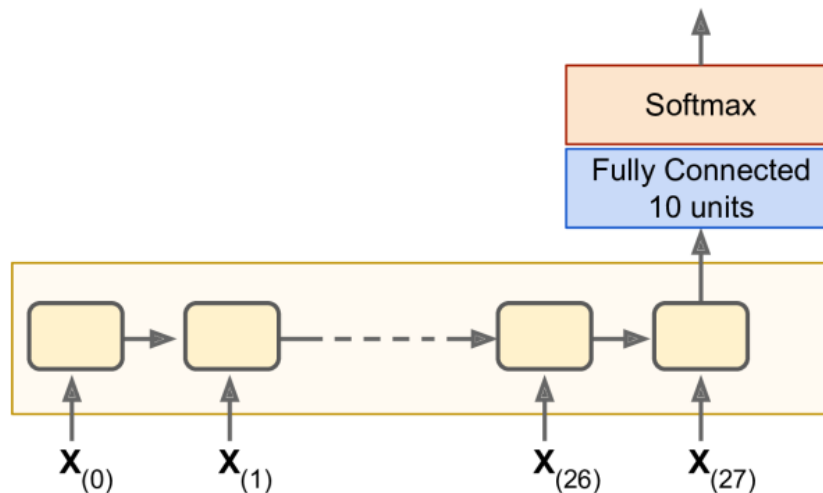


Figura 4.3. 8 : RNN cu layer dens și softmax; sursa: [10]

Merită menționat și conceptul de celulă de memorie, în special LSTM. Datorită faptului că ieșirile neuronilor sunt funcții de toate intrările lor din toate momentele de timp, se poate spune că aceștia au o formă de memorie, deci sunt celule de memorie. O celulă LSTM este o celulă de memorie mai complexă, care conține mai mulți neuroni, fiecare având funcții diferite. Celula LSTM este reprezentată în figura 4.3.6:

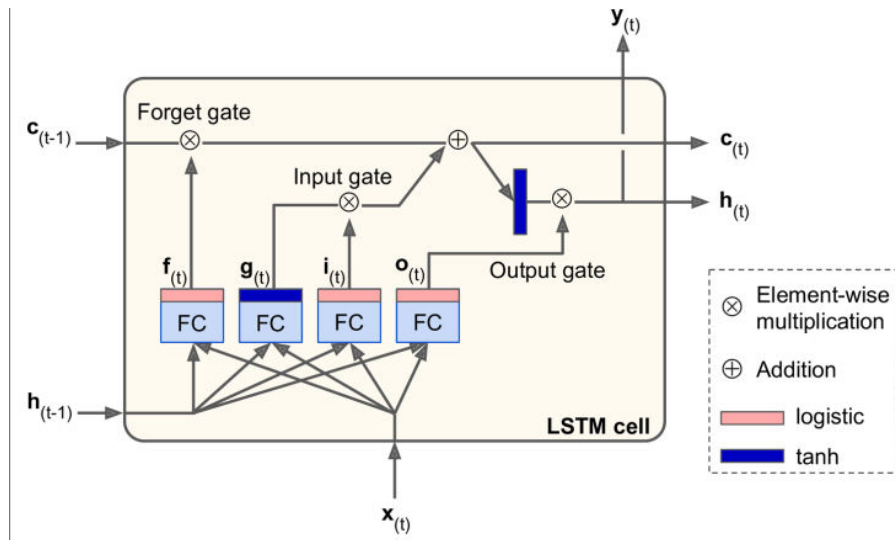
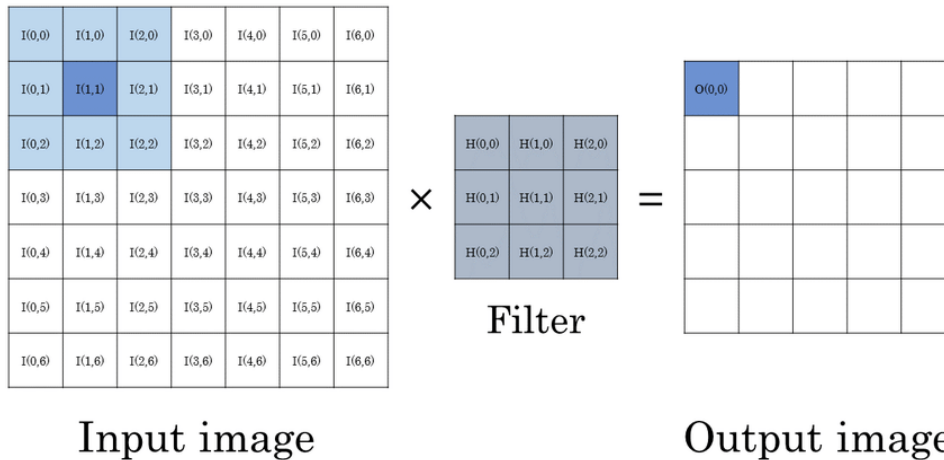


Figura 4.3. 9: Celulă LSTM; sursa: [10]

### 4.3.2 CNN

Rețelele neuronale bazate pe convoluție sunt cele mai potrivite pentru clasificarea de imagini. Arhitectura unor astfel de rețele constă în alternarea layerelor de convoluție cu layer de pooling, iar la final se adaugă un layer dens cu funcția de activare softmax.

Operația de convoluție constă în aplicarea (suprapunerea) unui filtru de convoluție peste o imagine, progresând pixel cu pixel și adunând valorile obținute prin înmulțirea valorilor din filtru cu cele din fereastră. Un filtru de convoluție are dimensiunile egale cu ale ferestrei de convoluție, așa ca în figura 4.3.7:


 Figura 4.3. 10: Convoluție; sursa: [www.researchgate.net](http://www.researchgate.net)

Layerelor de convoluție preiau matricea de intrare (în cazul primului layer de convoluție, aceasta este imaginea mfcc) și cu ajutorul unor filtre pătrate (de dimensiune 2x2, de exemplu), aplică convoluția pe intrare, mutând fereastra de convoluție pixel cu pixel (neuron cu neuron). Fiecare neuron din layerul de convoluție este conectat cu câte o zonă (fereastră) din input, iar valoarea obținută în urma aplicării convoluției pe zona

respectivă reprezintă intrarea neuronului. Structura și modul de funcționare al layerelor de convoluție pentru imagini de intrare cu un singur canal sunt ilustrate în figura 4.3.8, preluată din sursa [10]:

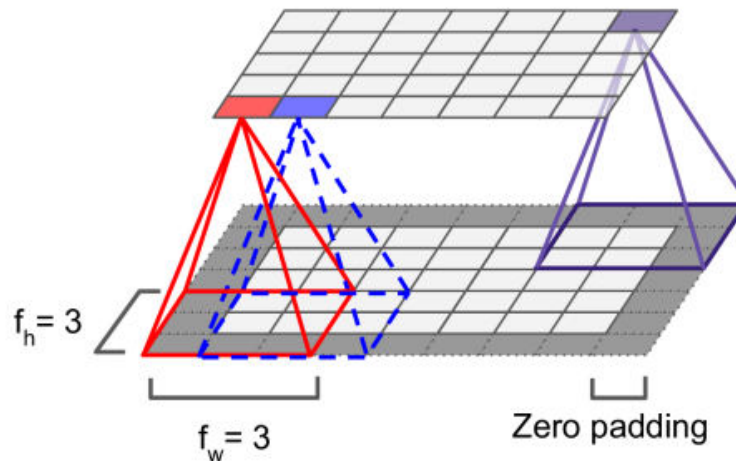


Figura 4.3. 11: Layer convoluțional

Din figură reiese faptul că am aplicat padding imaginii de intrare, adică am încadrat-o cu pixeli cu valoarea 0, pentru ca să se păstreze dimensiunea layerului următor egală cu cea a primului layer.

Filtrele de convoluție conțin valori dispuse în așa fel încât să evidențieze o anumită trăsătură (feature) a imaginii de intrare, astfel, aplicând filtre diferite unei imagini obținem ceea ce se numește “feature maps”. În figura 4.3.9 preluată din sursa [10] avem layere convoluționale legate între ele, cu mai multe feature maps și cu imagini cu 3 canale (RGB).



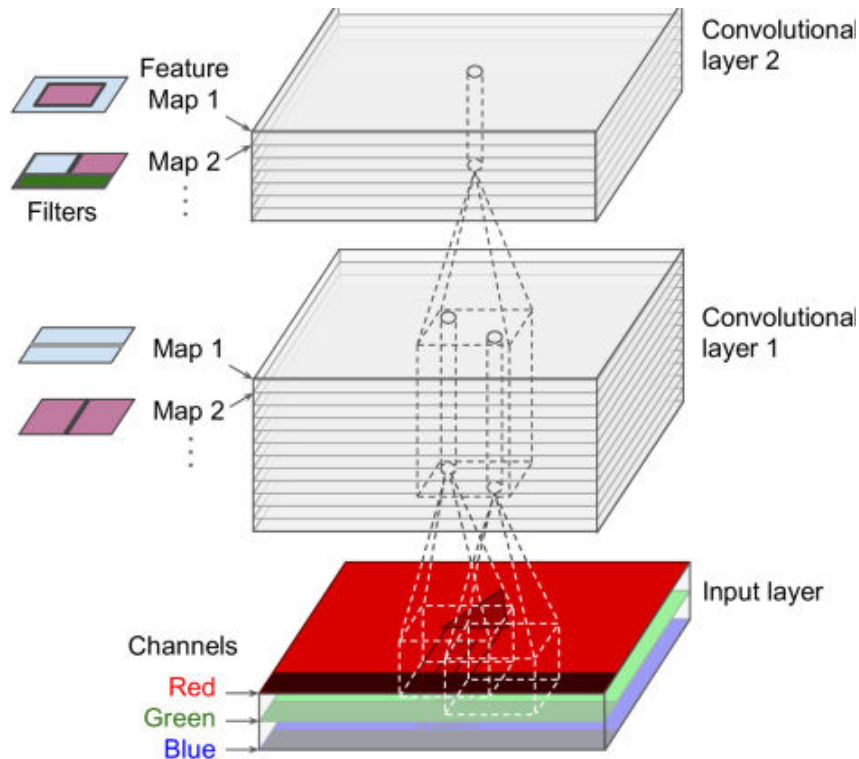


Figura 4.3. 12: Layere de convoluție cu mai multe feature maps și cu imagini RGB

Layerele de pooling sunt folosite pentru a reduce din intrare (pentru a micșora imaginea de intrare). Aceasta se face în scopul reducerii numărului de parametri (scăzând astfel posibilitatea de overfitting), dar și în scopul reducerii cantității de memorie și a puterii de procesare utilizate. Similar unui layer convoluțional, fiecare neuron din layerul de pooling este conectat la o zonă anume din intrarea lui, iar ieșirea neuronului din layerul de pooling depinde de tipul de layer de pooling. Dacă este un layer de tip “average”, atunci ieșirea va fi media valorilor din fereastra de intrare, iar dacă este un layer de tip “max”, ieșirea lui va fi valoarea maximă din fereastra de intrare. Astfel, în loc să continuăm propagarea prin rețea a 4 valori (în cazul în care fereastra este 2x2), vom continua cu o singură valoare, deci intrarea pentru următorul layer convoluțional va fi redusă ca dimensiune. În figura 4.3.10 preluată din sursa [10] avem ilustrat conceptul de pooling:

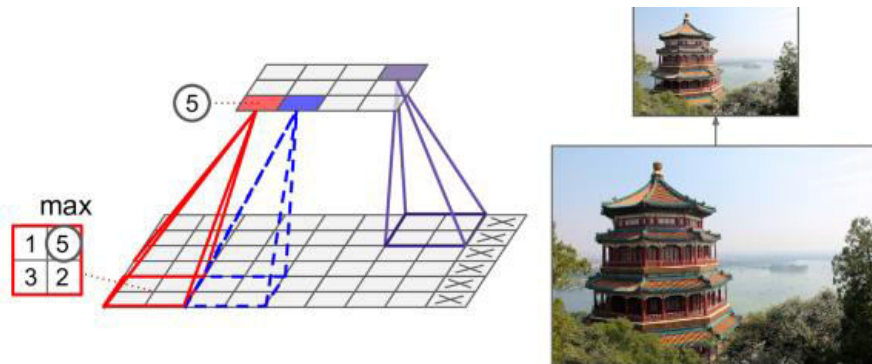




Figura 4.3. 13: Pooling layer

Rețelele convoluționale construite de mine în acest proiect au următoarea arhitectură: de 4 ori câte un layer de convoluție urmat de câte un layer max-pooling și de câte unul de dropout, apoi un layer dens cu funcția de activare ReLU (Rectified Linear Unit), iar în final un layer dens (cel de clasificare) cu funcția de activare softmax. Pe măsură ce intrăm în adâncimea rețelei, layerurile convoluționale folosesc un număr din ce în ce mai mare de filtre, deci se obțin din ce în ce mai multe feature maps, mai precis numărul acestora se dublează cu fiecare layer convoluțional. Așadar, începem cu 16 filtre și terminăm cu 128.

Layerurile de dropout au rolul de a preveni fenomenul de overfitting. Acest fenomen constă în extragerea unor informații/concluzii pe baza datelor de antrenament care sunt adevărate în interiorul setului de antrenament, dar sunt false în afara lui, deci vor face ca modelul să clasifice greșit datele noi. Dropout-ul oprește acest fenomen ignorând contribuțiile câtorva neuroni luați la întâmplare în timpul antrenamentului.

### 4.3.3 Transfer Learning

Transfer learning presupune folosirea unei rețele neuronale pre-antrenate pe un set de date pentru învățare. Layerurile din aceste rețele pre-antrenate pot fi blocate pentru învățare sau, cu opțiunea unfreeze, se pot deschide pentru a fi antrenate pe datele furnizate. La o rețea pre-antrenată se mai pot adăuga și alte layeruri, cum ar fi un layer de clasificare cu softmax.

### 4.3.4 MobileNet v2

MobileNet este un exemplu de rețea pre-antrenată pe setul de date ImageNet, care conține imagini cu 3 canale, de dimensiuni 224 x 224, imagini ale unor obiecte din WordNet (1000 de imagini pentru fiecare din cele 80000 de substantive). Această rețea a fost creată pentru a face posibilă rularea algoritmilor de machine learning pe dispozitivele mobile.

Rețeaua folosită în experimentele din acest proiect are 155 de layeruri, printre care layeruri convoluționale, layeruri de batch normalization și layeruri de convoluție depthwise.

Layerurile de batch normalization au rolul de a aduce toate intrările în același interval pentru a facilita învățarea. De exemplu, faptul că avem unele intrări în intervalul 0-1 și altele în intervalul 100-1000 ar îngreuna învățarea dacă nu ar fi prezent un layer de batch normalization.

Conform sursei [17], convoluția depthwise se bazează pe ideea înlocuirii unui operator convoluțional întreg cu o versiune factorizată care desparte convoluția în două layeruri separate. Primul layer face o filtrare ușoară aplicând câte un singur filtru convoluțional per canal de intrare. Al doilea layer face o convoluție 1x1 (per pixel), care ajută la găsirea de noi trăsături (features) prin calcularea de combinații liniare între canalele de intrare.

Arhitectura rețelei MobileNet v2 conține mai multe blocuri formate din layerurile amintite mai devreme (building blocks), și este asemănătoare celei din fig.4.3.11:

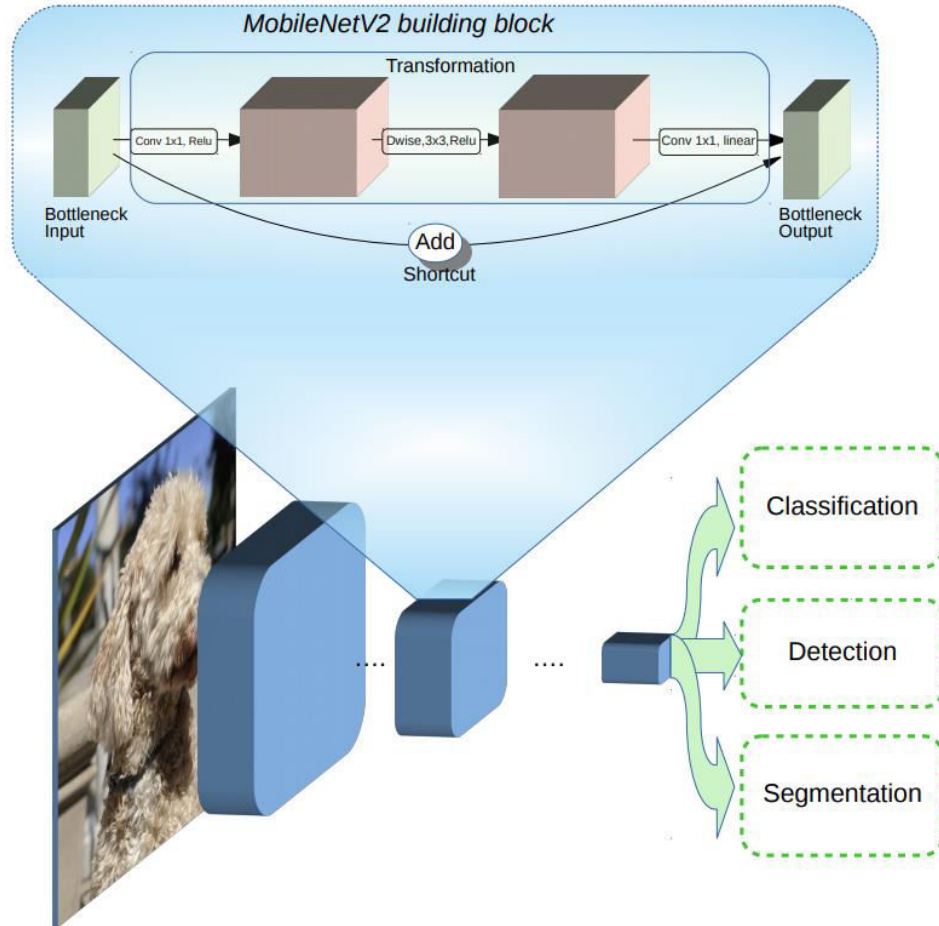


Figura 4.3. 14: MobileNet v2

MobileNet v2, varianta cu imagini de dimensiune 224x224, a ajuns la o acuratețe de 74.7% la validare pe setul de date ImageNet.

#### 4.3.5 DenseNet

Rețeaua DenseNet, după cum îi spune și numele, se bazează pe blocuri dense. Un bloc dens este o serie de layere în care fiecare layer este conectat direct cu toate celelalte. Astfel, fiecare layer din blocul dens primește feature-maps-urile tuturor blocurilor dinaintea lui și înaintează layerelor următoare feature maps-urile lui (deci blocul are o natură feed-forward). Conform sursei [8], faptul că toate layerele dintr-un bloc sunt conectate între ele împiedică pierderea de informație despre intrări pe măsura înaintării în adâncimea rețelei. În figura 4.3.12 avem un exemplu de bloc dens cu 5 layere, preluat din sursa [8]:

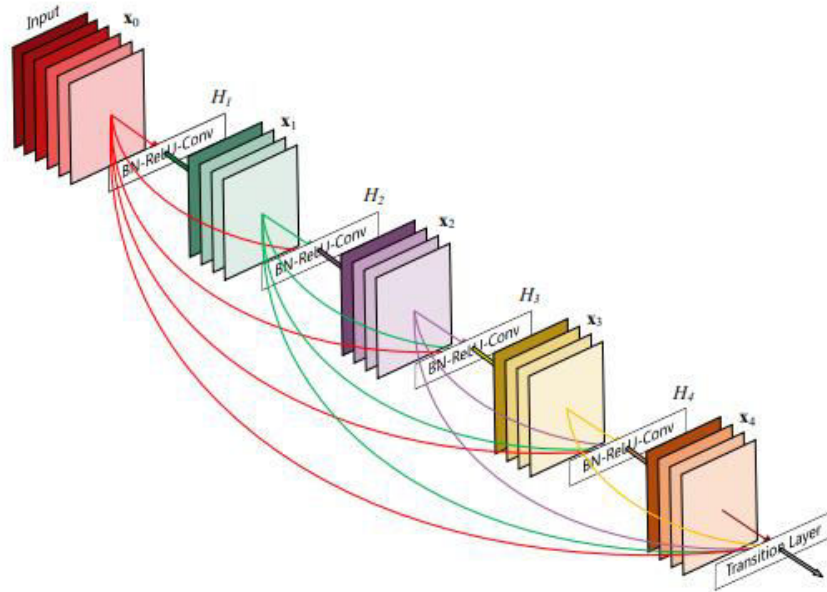


Figura 4.3. 15: Bloc dens cu 5 layere

Layerete DenseNet sunt foarte înguste (12 feature maps per layer), adăugând numai un set mic de feature maps la setul de cunoștințe generale al rețelei și păstrând neschimbate restul de feature maps. Pe lângă faptul că au mai puțini parametri, aceste rețele sunt mai ușor de antrenat deoarece informația curge mult mai bine prin layere<sup>[8]</sup>. De asemenea, conexiunile dense creează multe legături scurte în rețea, care au un efect de regularizare și reduc fenomenul de overfitting pe seturi de date mai mici<sup>[8]</sup>.

Arhitectura rețelei DenseNet este asemănătoare celei din figura 4.3.13, preluată din [8]:

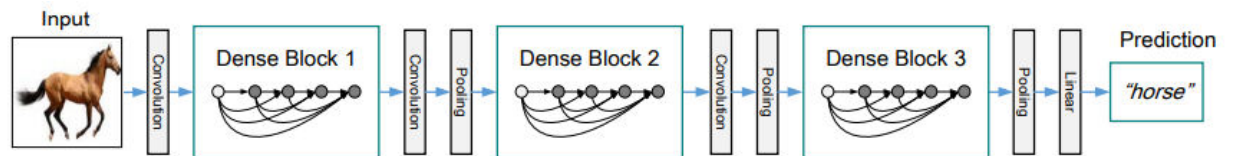


Figura 4.3. 16: Arhitectură DenseNet

Layerete dintre cele 3 blocuri Dense se numesc layere de tranziție și au rolul de a modifica mărimea feature map-urilor prin convoluție și pooling<sup>[8]</sup>.

Pe setul de date ImageNet, DenseNet-121 (variantea folosită în proiect) a obținut o acuratețe la validare de 74.98%<sup>[1]</sup>, mai mare decât cea a MobileNetV2.

#### 4.3.6 Comparăție între MobileNet și DenseNet

Experimental, am observat că pe setul de date VoxCeleb utilizat în acest proiect am obținut rezultate mult mai bune cu DenseNet decât cu MobileNet. Acest lucru se datorează faptului că conceptul de layere dense face posibilă utilizarea a mai multe layere (DenseNet 121 are peste 400 de layere, iar MobileNetV2 are 155) fără ca numărul de parametri să crească. Legăturile directe între layere fac ca fiecare layer să aibă acces la

baza de cunoștințe a layerelor anterioare și facilitează astfel trecerea prin rețea a informației acumulate.

Cu toate că abordarea bazată pe DenseNet a dat rezultate mai bune decât cea bazată pe MobileNet, ambele abordări au fost depășite de cea cu o rețea convoluțională simplă cu 3 layere. Acest lucru s-a întâmplat din cauza faptului că aceste două rețele pre-antrenate au fost antrenate pe imagini care diferă foarte mult față de mfcc-urile și melspectrogramele utilizate în acest proiect. Imaginile pe care au fost antrenate aceste două rețele înfățișează animale sau diverse obiecte, de aceea ele diferă de mfcc-uri și melspectrograme cel puțin din punctul de vedere al entropiei, mărime pe care o voi introduce într-un capitol următor.

## Capitolul 5. Proiectare de Detaliu si Implementare

Proiectul este organizat în mai multe module, fiecare fiind responsabil de anumite etape ale flow-ului. Etapele principale ale flow-ului recunoașterii de vorbitor sunt:

1. extragerea de features din clipurile audio din dataset (într-o structură pandas dataframe)
2. împărțirea dataframe-ului în setul de date de train și setul de date de test
3. construirea modelului
4. compilarea modelului
5. antrenarea modelului
6. evaluarea modelului folosind metricile explicate în capitolul 2
7. furnizarea către model a unor clipuri străine și detectarea vorbitorilor din acestea

Cele 4 module principale ale proiectului, deci, se numesc `signal_processing`, `data_load`, `learn` și `evaluate`. Opțional, am creat și un modul `render` cu ajutorul căruia putem afișa imaginile extrase din clipuri. În continuare, voi intra în detaliu legat de fiecare componentă a proiectului, trecând prin fiecare dintre etapele principale descrise mai sus.

### 5.1 Extragerea de features din clipurile audio

Pentru aceasta, avem modulele `signal_processing` și `data_load`. Modulul `data_load` are ca și funcții principale funcțiile de tip `make_dataframe`, care apelează funcțiile de tip `extract_feature` din `signal_processing`. În funcțiile de tip `extract_feature` am folosit biblioteca de python `librosa`, care conține funcțiile `feature.mfcc` și `feature.melspectrogram`. În funcția `extract_features_mfcc_seconds`, dată ca exemplu mai jos, am extras cu ajutorul funcției `librosa.load` clipul audio din fișierul cu calea furnizată, apoi am obținut imaginea de tip `mfcc` cu funcția `librosa.feature.mfcc`. Pentru rețelele convoluționale este important ca imaginile de intrare să aibă aceleași dimensiuni, de aceea am folosit funcția `numpy.pad` pentru a adăuga biți de 0 imaginilor mai mici.

```
def extract_features_mfcc_seconds(file_name, nr_mfccs,
max_pad_len, seconds):
    audio, sample_rate = librosa.load(file_name,
res_type='kaiser_fast', duration = seconds)
    mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate,
n_mfcc=nr_mfccs)    pad_width = max_pad_len -
mfccs.shape[1]
    mfccs = np.pad(mfccs, pad_width=((0, 0), (0,
pad_width)), mode='constant')
    return mfccs
```

În afară de această funcție, în modulul `signal_processing` mai există și alte funcții cu ajutorul cărora putem extrage imagini-features dintr-un clip audio, și anume: `extract_features_mfcc` (care nu primește ca parametru un număr de secunde, funcție folosită în experimentele inițiale) `extract_features_mfcc_htk` (extrage `mfcc` după metoda `htk`), `extract_features_mfcc_sr` (primește ca parametru rata de eșantionare a clipului

audio) și, funcția care va fi ilustrată în exemplul de cod următor, `extract_features_melspectrogram`:

```
def extract_features_melspectrogram(file_name, nr_fft,
nr_mels, hop_length, max_pad_len, seconds):
    audio, sample_rate = librosa.load(file_name,
res_type='kaiser_fast', duration=seconds)
    mel = librosa.feature.melspectrogram(y=audio,
sr=sample_rate, n_fft=nr_fft, hop_length=hop_length,
n_mels=nr_mels)
    pad_width = max_pad_len - mel.shape[1]
    mel = np.pad(mel, pad_width=((0, 0), (0,
pad_width))), mode='constant')
    return mel
```

Această funcție diferă de celelalte prin faptul că utilizează funcția `librosa.feature.melspectrogram` în loc de `librosa.feature.mfcc`. Parametrii funcției `librosa.feature.melspectrogram` care diferă de `mfcc` și merită menționați sunt: `n_fft` (lungimea ferestrei pe care se aplică transformata Fourier), `hop_length` (numărul de eșantioane dintre două ferestre succesive) și `n_mels` (înălțimea imaginii).

În modulul `data_load` avem funcțiile responsabile de crearea dataframe-urilor `pandas` care sunt niște tabele în care avem array-ul-imagine (coloana `feature`), numele clasei (numele vorbitorului, coloana `class_label`) și, pentru cazurile în care folosim `class weights` în antrenamentul modelului, numărul clasei (coloana `class_no`). Acest număr este util în momentul în care se apelează funcția `numpy.bincount` în calcularea `class weights`-urilor (pentru numărarea `sample`-urilor din fiecare clasă este nevoie ca clasa să fie exprimată sub forma unui număr întreg). Ca exemplu, ilustrăm în secvența de cod care urmează funcția `make_dataframe_class_no` din modulul `data_load`:

```
def make_dataframe_class_no(nr_speakers_total, base_path,
nr_mfccs, max_pad_len, nr_seconds):
    metadata = pd.read_csv('vox1_meta.csv')
    features = []
    # Iterate through each sound file and extract the
features
    nr_of_speakers = 0
    for index, row in metadata.iterrows():
        speaker_dir = os.path.join(base_path,
str(row["ID"]))
        entries = os.scandir(speaker_dir)

        for folder in entries:
            files = os.scandir(folder)
            for f in files:
                file_name =
os.path.join(str(speaker_dir), str(folder.name),
os.path.basename(f))
                class_label = row["Class_name"]
```

```

                                classNo =
class_label_to_nr(class_label)
                                data =
signal_processing.extract_features_mfcc_seconds(file_name,
nr_mfccs, max_pad_len, nr_seconds)
                                features.append([data,
class_label, classNo])
                                if nr_of_speakers == nr_speakers_total - 1:
                                    break
                                else:
                                    nr_of_speakers += 1
                                # Convert into a Pandas dataframe
                                featuresdf = pd.DataFrame(features,
columns=['feature', 'class_label', 'class'])
                                print('Finished feature extraction from ',
len(featuresdf), ' files')
                                return featuresdf

```

Această funcție iterează prin folderele din setul de date și, pe baza informațiilor din fișierul `vox1_meta` extrage mfcc-urile din fiecare clip și le adaugă în tabloul `features`, alături de numele clasei și de numărul clasei. Conversia între nume și număr de clasă (și invers) se face cu ajutorul funcțiilor `class_label_to_nr` și `nr_to_class_label` din modulul `data_load`. La final, tabloul `features` este convertit într-un dataframe pandas, care este returnat.

În modulul `data_load` mai avem funcții asemănătoare, care creează dataframe-uri, cum ar fi `make_dataframe_melspectrogram` și `make_dataframe_2speakers` (funcție care extrage `features` numai din clipurile aparținând a două persoane, numele acestora fiind date ca parametri).

## 5.2 Împărțirea dataframe-ului în setul de date de train și setul de date de test

Pentru aceasta, folosim funcția `make_train_test_sets` din modulul `data_load`. În această funcție, din dataframe-ul dat ca parametru se extrag feature-urile și numele claselor (labels) sub formă de numpy arrays. Apoi, numele claselor sunt encodeate cu `LabelEncoder`, iar cu funcția `sklearn.model_selection.train_test_split` feature-urile și etichetele sunt împărțite în seturile de train și test, 20% din date mergând în setul de test. La final, imaginile (`x_train` și `x_test`) sunt redimensionate pentru a putea fi folosite ca intrări la rețeaua convoluțională. Funcția `make_train_test_sets` arată în felul următor:

```

def make_train_test_sets(featuresdf, num_rows, num_columns,
num_channels):
    # Convert features and corresponding classification
labels into numpy arrays
    X = np.array(featuresdf.feature.tolist())
    y = np.array(featuresdf.class_label.tolist())

    # Encode the classification labels

```

```
le = LabelEncoder()
yy = to_categorical(le.fit_transform(y))

# split the dataset

x_train, x_test, y_train, y_test = train_test_split(X,
yy, test_size=0.2, random_state = 42)
num_labels = yy.shape[1]

if num_channels == 3:
    x_train = x_train.reshape(x_train.shape[0],
num_rows, num_columns)
    x_test = x_test.reshape(x_test.shape[0],
num_rows, num_columns)

    x_train = np.repeat(x_train[... , np.newaxis], 3,
-1)
    x_test = np.repeat(x_test[... , np.newaxis], 3, -
1)

    result_sets = [x_train, x_test, y_train, y_test,
num_labels]
    return result_sets

x_train = x_train.reshape(x_train.shape[0], num_rows,
num_columns, 1)
x_test = x_test.reshape(x_test.shape[0], num_rows,
num_columns, 1)
result_sets = [x_train, x_test, y_train, y_test,
num_labels]
return result_sets
```

### 5.3 Construirea modelului

În modulul `learn`, avem mai multe funcții de construire a modelelor: `build_CNN_model`, `build_regularized_CNN_model`, `create_model_with_two_inputs`, `build_model_mobilenet` și `build_model_densenet`. Funcțiile care construiesc modele CNN iau ca parametri dimensiunile imaginilor de intrare și numărul de clase (`num_labels`). Numărul de canale nu este dat ca parametru, deoarece la folosirea rețelelor convoluționale am lucrat doar cu imagini cu un canal. În exemplul următor de cod avem funcția `build_CNN_model`, care creează un model convoluțional cu arhitectura descrisă în capitolul anterior.

```
def build_CNN_model(num_rows, num_columns, num_labels):
    num_channels = 1
    filter_size = 2

    # Construct model
```



```
model = Sequential()
model.add(Conv2D(filters=16, kernel_size=2,
input_shape=(num_rows, num_columns, num_channels),
activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))

model.add(Conv2D(filters=32, kernel_size=2,
activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))

model.add(Conv2D(filters=64, kernel_size=2,
activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))

model.add(Conv2D(filters=128, kernel_size=2,
activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))
model.add(GlobalAveragePooling2D())

model.add(Dense(num_labels, activation='softmax'))
return model
```

Funcțiile de creare a modelelor cu rețea pre-antrenată diferă de cele cu CNN simplu prin faptul că iau ca parametru modelul pre-antrenat (`base_model`, obținut prin apelul funcțiilor `get_mobilenet`, respectiv `get_densenet` din modulul `learn`) și numărul de clase (`num_labels`). Modelului pre-antrenat (din care se lasă deoparte layerul propriu de clasificare, cu `include_top=False`) i se adaugă un layer de clasificare cu softmax, rezultatul final fiind un model de tip `Sequential`:

```
model = tf.keras.Sequential([
    base_model,
    Dense(num_labels, activation='softmax')
])
```

Merită menționată și funcția de construire a modelului cu două input-uri, și anume imagini de tip mfcc și melspectrograme. Această funcție a fost realizată asemănător celei din tutorialul tensorflow din sursa [3]. Modelul rezultat a fost desenat cu funcția `plot_model` și este reprezentat în figura 5.3.1. Pipeline-ul este același pentru ambele intrări, iar la final, după pooling, ieșirile sunt concatenate.

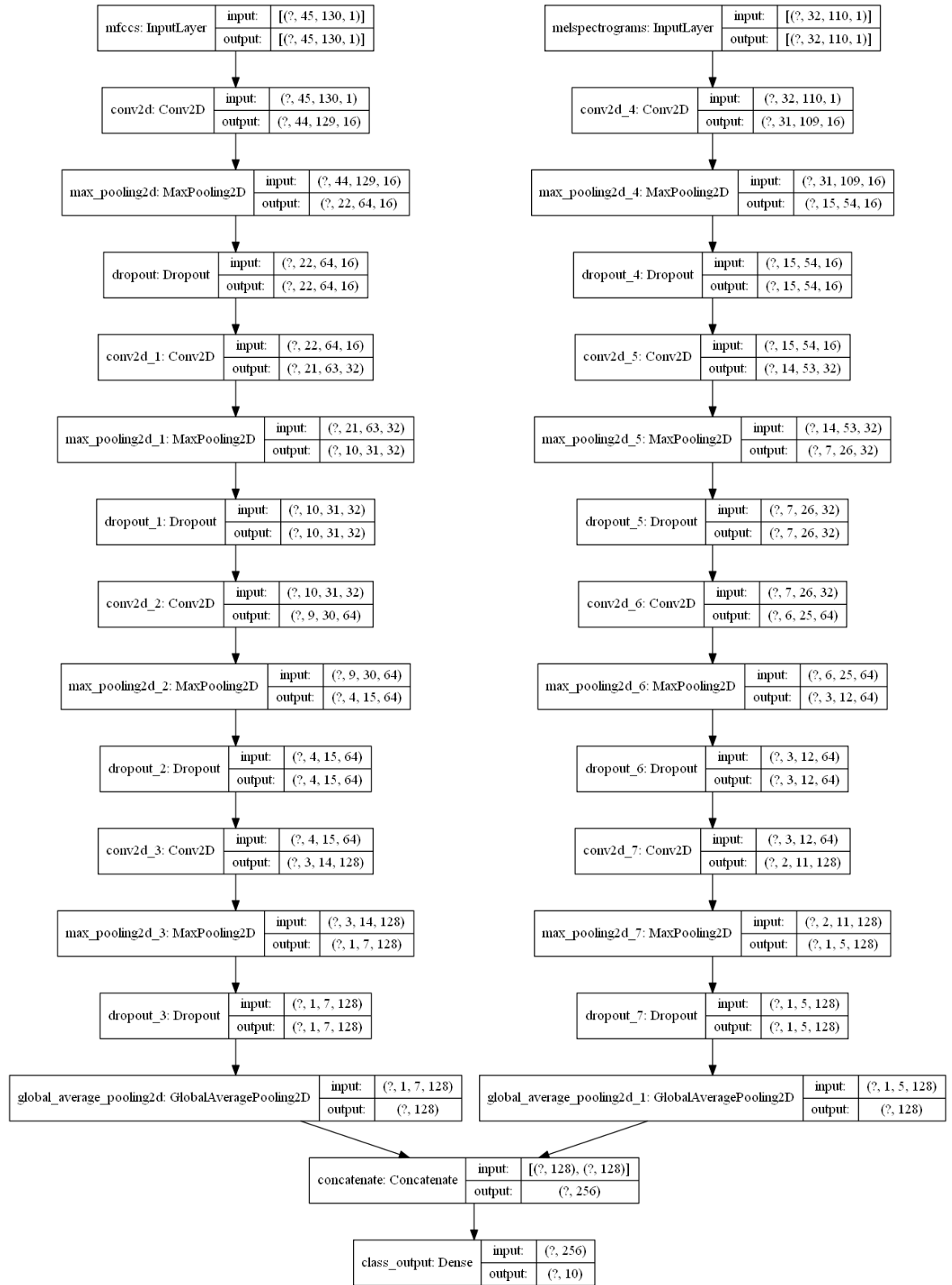


Figura 5.3. 1: Modelul cu două intrări

## 5.4 Compilarea modelului

Pentru compilarea modelelor avem două funcții în modulul `learn`, `compile` și `compile_model_pretrained_net`. Motivul pentru care am creat două funcții este că la compilarea modelelor cu rețea pretrained am pasat ca parametru și un learning rate. Ambele funcții menționate apelează funcția `compile` din `tensorflow.keras`:

```
def compile(model):  
    # Compile the model  
    model.compile(loss='categorical_crossentropy',  
metrics=['accuracy'], optimizer='adam')
```

## 5.5 Antrenarea modelului

Funcțiile necesare pentru antrenarea modelelor se află tot în modulul `learn` și se numesc `train_model`, `train_model_class_weights`, respectiv `train_model_two_inputs`. Toate aceste funcții se bazează pe funcția `model.fit` din `tensorflow`. Cu ajutorul unui `ModelCheckpoint`, modelul (weight-urile acestuia) cel mai bun din punct de vedere al acurateței este salvat în folderul `saved_models`, cu numele `weights.best`. numele dat ca parametru (`model_name`). În exemplul de cod următor avem funcția `train_model_class_weights`, care primește ca parametri:

- modelul compilat, `model`
- datele de antrenament și de test obținute din funcția `make_train_test_sets`, `train_test_data`
- mărimea batch-size-ului, `num_batch_size`
- numele modelului sub care va fi salvat, `model_name`
- numele folderului principal unde vor fi salvate graficele care arată cum evoluează loss-ul și acuratețea (pentru a fi afișate în `tensorboard`), `logdir_name`, și numele folderului din interiorul folderului cu `logdir_name`, unde se salvează efectiv datele, `name`. `Name` poate fi `default` sau `regularization`.
- `class_weight`, un array asociativ obținut cu ajutorul funcției `calculate_class_weights` din modulul `learn`. Acest dicționar conține ponderile pentru fiecare clasă, mai mari pentru clasele cu exemple mai puține, și mai mici pentru clasele cu exemple mai multe.

```
def train_model_class_weights(model, train_test_data,  
num_epochs, num_batch_size, model_name, name, logdir_name,  
class_weight):  
    saved_models = pathlib.Path('saved_models/')  
    if not saved_models.exists():  
        os.makedirs(saved_models)  
  
    checkpointer =  
ModelCheckpoint(filepath='saved_models/weights.best.' +  
model_name + '.hdf5', verbose=1, save_best_only=True)
```

```
x_train = train_test_data[0]
x_test = train_test_data[1]
y_train = train_test_data[2]
y_test = train_test_data[3]

start = datetime.now()

logdir = make_logdir(logdir_name)

history = model.fit(x_train, y_train,
batch_size=num_batch_size, epochs=num_epochs,
validation_data=(x_test, y_test), callbacks=[checkpointer,
tf.keras.callbacks.TensorBoard(logdir/name)],
verbose=1, class_weight=class_weight)
duration = datetime.now() - start
print("Training completed in time: ", duration)
return history
```

Această funcție returnează un obiect de tip history (returnat de funcția `model_fit`, apelată în interior). Obiectul history va fi util pentru a afișa cu `pyplot` evoluția acurateței și a loss-ului pe parcursul epocilor.

## 5.6 Evaluarea modelului folosind metricile explicate în capitolul 2

În modulul `evaluate`, avem mai multe funcții pentru evaluarea modelelor obținute, folosind metricile din capitolul 2 (acuratețe, precizie, recall și afișarea matricei de confuzie).

Cele 3 modalități principale de evaluare sunt realizate prin funcțiile `display_metrics`, `evaluate_model` și `plot_history`. De asemenea, în modulul `evaluate` mai avem și funcția `plot_compare_val_loss`, care compară două modele din punct de vedere al loss-ului din validare, pe parcursul epocilor de antrenament.

### 5.6.1 *display\_metrics*

Funcția `display_metrics` se folosește de `classification_report` și `confusion_matrix` din `sklearn.metrics`. Aceasta ia ca parametri modelul, datele de test și numele sub care au fost salvate weight-urile modelului, pentru a-l încărca de acolo și a-l evalua.

```
def display_metrics(model, model_name, train_test_data):
    x_train = train_test_data[0]
    x_test = train_test_data[1]
    y_train = train_test_data[2]
    y_test = train_test_data[3]

    model.load_weights("saved_models/weights.best." +
model_name + ".hdf5")
```

```

y_pred = model.predict_classes(x_test, batch_size=12,
verbose=0)
y_true = np.argmax(y_test, axis=1)

print(classification_report(y_true, y_pred))
print("Confusion matrix: ")
print(confusion_matrix(y_true, y_pred))

```

Un exemplu de rulare al acestei funcții este regăsit în figura 5.6.1:

```
[ ] evaluate.display_metrics(model, 'CNNCw', result_sets)
```

	precision	recall	f1-score	support
0	0.84	1.00	0.91	16
1	1.00	1.00	1.00	18
2	1.00	1.00	1.00	43
3	0.94	0.85	0.89	20
4	1.00	0.54	0.70	13
5	0.84	0.90	0.87	30
6	0.88	1.00	0.93	14
7	1.00	0.88	0.93	16
8	0.83	0.83	0.83	12
9	0.73	0.92	0.81	12
accuracy			0.91	194
macro avg	0.91	0.89	0.89	194
weighted avg	0.92	0.91	0.91	194

Confusion matrix:

```

[[16  0  0  0  0  0  0  0  0  0]
 [ 0 18  0  0  0  0  0  0  0  0]
 [ 0  0 43  0  0  0  0  0  0  0]
 [ 1  0  0 17  0  2  0  0  0  0]
 [ 2  0  0  0  7  0  1  0  0  3]
 [ 0  0  0  0  0 27  1  0  2  0]
 [ 0  0  0  0  0  0 14  0  0  0]
 [ 0  0  0  1  0  1  0 14  0  0]
 [ 0  0  0  0  0  1  0  0 10  1]
 [ 0  0  0  0  0  1  0  0  0 11]]

```

Figura 5.6. 1: Classification report și confusion matrix

### 5.6.2 *evaluate\_model*

Această funcție afișează acuratețea modelului dat ca parametru, la antrenament și la testare. Pentru a calcula aceste valori, este nevoie ca datele de antrenament și de test să fie pasate ca parametri funcției. Funcția arată în felul următor:

```

def evaluate_model(model, model_name, train_test_data):
    x_train = train_test_data[0]
    x_test = train_test_data[1]

```

```

y_train = train_test_data[2]
y_test = train_test_data[3]
# Evaluating the model on the training and testing set
model.load_weights("saved_models/weights.best." +
model_name + ".hdf5")
score = model.evaluate(x_train, y_train, verbose=0)
print("Training Accuracy: ", score[1])

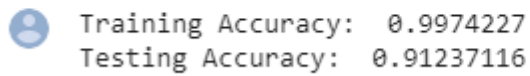
score = model.evaluate(x_test, y_test, verbose=0)
print("Testing Accuracy: ", score[1])

```

Pentru a obține valorile căutate, funcția apelează `model.evaluate` din tensorflow.

Rezultatele afișate de această funcție arată asemănător celor din figura 5.6.2:

```
[ ] evaluate.evaluate_model(model, 'CNNCw', result_sets)
```



Training Accuracy: 0.9974227  
Testing Accuracy: 0.91237116

Figura 5.6. 2: `evaluate_model`

### 5.6.3 *plot\_history*

Funcția `plot_history` are rolul de a afișa un grafic cu evoluția lossului la validare în funcție de epoci, și un grafic cu evoluția acurateței, tot în funcție de epoci. Următorul exemplu de cod ilustrează funcția:

```

def plot_history(history):
    fig, axs = plt.subplots(1, 2, figsize=(12, 5))#,
sharey=True)
    #axs[0].bar(names, values)
    #axs[1].scatter(names, values)
    #axs[2].plot(names, values)

    loss = history.history['loss']
    acc = history.history['accuracy']
    val_loss = history.history['val_loss']
    val_acc = history.history['val_accuracy']

    epochs = range(1, len(acc) + 1)

    axs[0].plot(epochs, loss, 'b', label='Training loss')
    axs[0].plot(epochs, val_loss, '--', label='Validation
loss')
    axs[0].set_title('Training and validation loss')
    axs[0].set_xlabel('Epochs')
    axs[0].set_ylabel('Loss')
    axs[0].legend(loc="upper right")

    axs[1].plot(epochs, acc, 'b', label='Training acc')

```

```

    axs[1].plot(epochs, val_acc, '--', label='Validation
acc')
    axs[1].set_title('Training and validation acc')
    axs[1].set_xlabel('Epochs')
    axs[1].set_ylabel('Loss')
    axs[1].legend(loc="upper right")

```

Pachetul cel mai important în această funcție este `matplotlib.pyplot`, cu ajutorul căruia s-au afișat graficele. În figura 5.6.3 avem un rezultat al aplicării acestei funcții:

```
[ ] evaluate.plot_history(history)
```

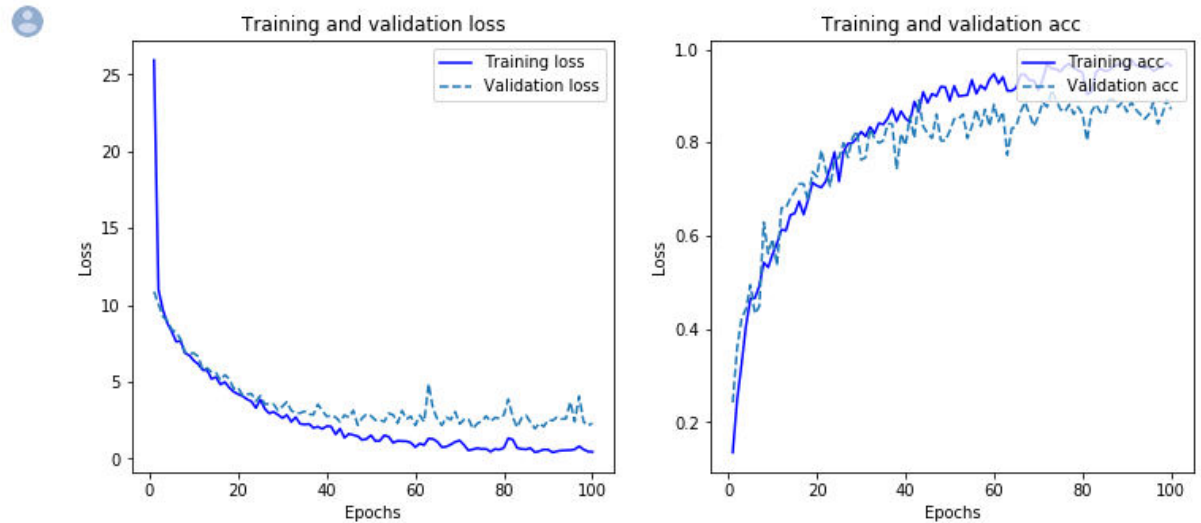


Figura 5.6. 3: `plot_history`

#### 5.6.4 `plot_compare_val_loss`

Funcția `plot_compare_val_loss` are rolul de a afișa într-un grafic comparativ valorile `loss`-ului în funcție de epocă, pentru două antrenări de modele. În proiect am folosit această funcție pentru a observa diferența pe care o face aplicarea regularizării L2 pe aceeași arhitectură de rețea convoluțională. În figura 5.6.4 avem rezultatul acestei funcții pentru modelul cu class weights din experimentul CNNCw, antrenat pe 100 de epoci:

```
[ ] evaluate.plot_compared_val_loss(history, reg_history, 'default', 'regularization L2')
```

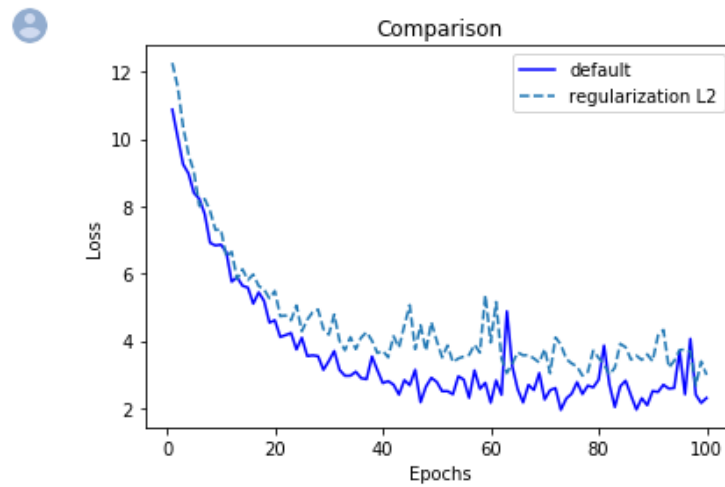


Figura 5.6. 4: plot\_compared\_val\_loss

### 5.7 Furnizarea către model a unor clipuri străine și detectarea vorbitorilor din acestea

Pentru a detecta vorbitorii avem funcția `print_prediction`, ilustrată în exemplul de cod următor:

```
def print_prediction(file_name, model, num_rows,
                    num_columns, num_channels, num_seconds):
    # Convert features and corresponding classification
    labels into numpy arrays
    y = np.array(['A.J._Buckley', 'A.R._Rahman',
                  'Aamir_Khan', 'Aaron_Tveit', 'Aaron_Yoo', 'Abbie_Cornish',
                  'Abigail_Breslin', 'Abigail_Spencer', 'Adam_Beach',
                  'Adam_Brody'])

    # Encode the classification labels
    le = LabelEncoder()
    yy = to_categorical(le.fit_transform(y))
    prediction_feature =
    signal_processing.extract_features_mfcc_seconds(file_name,
                                                    num_rows, num_columns, num_seconds)
    prediction_feature = prediction_feature.reshape(1,
                                                    num_rows, num_columns, num_channels)

    predicted_vector =
    model.predict_classes(prediction_feature)
    predicted_class =
    le.inverse_transform(predicted_vector)
    print("The predicted class is:", predicted_class[0],
          '\n')
```



```
predicted_proba_vector =  
model.predict_proba(prediction_feature)  
predicted_proba = predicted_proba_vector[0]  
for i in range(len(predicted_proba)):  
    category = le.inverse_transform(np.array([i]))  
    print(category[0], "\t\t : ",  
format(predicted_proba[i], '.32f') )
```

Această funcție preia ca parametru calea spre un fișier .wav și, cu funcția `model.predict_proba` din tensorflow obține un vector cu probabilitățile ca exemplul să aparțină fiecărei clase. Apoi, cu `LabelEncoder`-ul `le`, transformă clasele encodeate one-hot în nume de clase (labels), cu funcția `le.inverse_transform`.

Așadar, am detaliat cele mai importante funcții din module, parcurgând mersul logic al recunoașterii de vorbitor, de la citirea datelor de intrare și extragerea de features, până la detecția unui vorbitor dintr-un clip nou.

În continuare, voi prezenta o listă succintă a funcțiilor din proiect care nu au fost detaliate mai sus, grupate pe module:

- modulul `render`: `show_mfccs` (afișează imaginea mfcc extrasă din clipul din calea pasată ca parametru)
- modulul `learn`: `get_mobilenet`, `get_densenet` (returnează modelele pre-antrenate), `make_logdir` (creează directorul `tensorboard_logs`, funcție utilitară apelată de alte funcții din modul)
- modulul `evaluate`: `evaluate_before_training` (evaluează modelul pasat ca parametru pe datele de test furnizate, înainte de antrenament), `evaluate_model_two_inputs`, `display_metrics_two_inputs` (la fel ca și `evaluate_model` și `display_metrics`, doar că pentru modelul cu două intrări)

## Capitolul 6. Testare și Validare

Ca și metodă de testare a modulelor acestui proiect am realizat mai multe experimente în Jupyter notebooks, ale căror mod de lucru și rezultate le voi prezenta în continuare.

### 6.1 CNN1

Primul experiment din această serie este rețeaua convoluțională numită CNN1. Pentru a realiza experimentul am creat o rețea neuronală convoluțională cu arhitectura stabilită anterior și am antrenat-o timp de 10 epoci. Numărul claselor este 2 (primii 2 vorbitori, ambii bărbați, A.J. Buckley, și A.R. Rahman), metoda aleasă pentru extragerea de features este mfcc, iar dimensiunile imaginilor sunt de 45 x 2900 (am ales 45 pentru că am văzut în experimentele altora că este o valoare uzuală pentru lățimea imaginii, iar 2900 a fost ales pentru a cuprinde conținutul celui mai lung dintre clipuri). Pentru a ajunge la lungimea de 2900, imaginilor mai mici li s-a aplicat padding. Batch size-ul în acest experiment a fost de 121.

În urma efectuării acestui prim experiment am obținut 95% acuratețe la testare, precizia, recall-ul și matricea de confuzie arătând ca în figura 6.1:

```
In [10]: evaluate.display_metrics(model, result_sets)
```

	precision	recall	f1-score	support
0	0.87	1.00	0.93	13
1	1.00	0.93	0.96	29
accuracy			0.95	42
macro avg	0.93	0.97	0.95	42
weighted avg	0.96	0.95	0.95	42

```
Confusion matrix:
[[13  0]
 [ 2 27]]
```

Figura 6. 1: Rezultate pentru CNN1

Evoluția loss-ului și a acurateței pe parcursul epocilor a fost ilustrată cu funcția `evaluate.plot_history`, și rezultatul arată ca în figura 6.2:

```
In [11]: evaluate.plot_history(history)
```

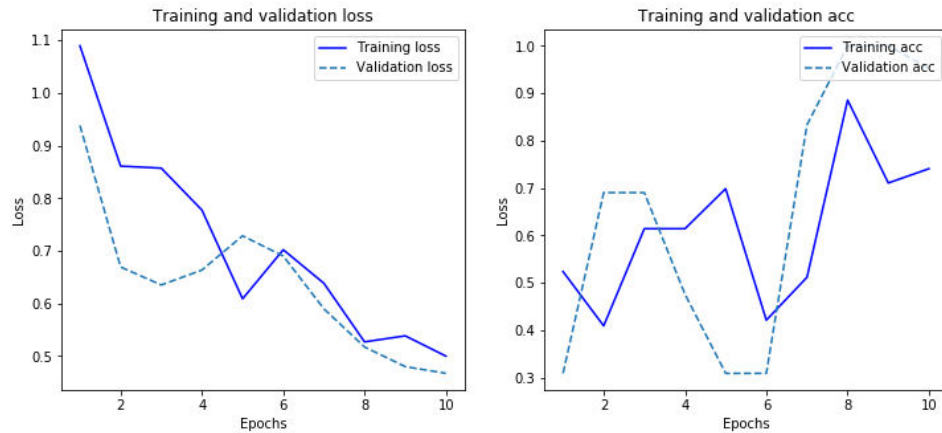


Figura 6. 2: Evoluția acurateței și a loss-ului pentru CNN1

După antrenarea acestui model, tot în cadrul primului experiment, am antrenat un al doilea model, similar cu primul, dar căruia i-am adăugat înainte de layerul de clasificare un layer dens cu funcția de activare ReLU și cu regularizare L1. Loss-ul pentru acest model regularizat a fost clar mai mare, după cum se vede în figura 6.3:

```
In [14]: evaluate.plot_compared_val_loss(history, reg_history, 'default', 'regularization L2')
```

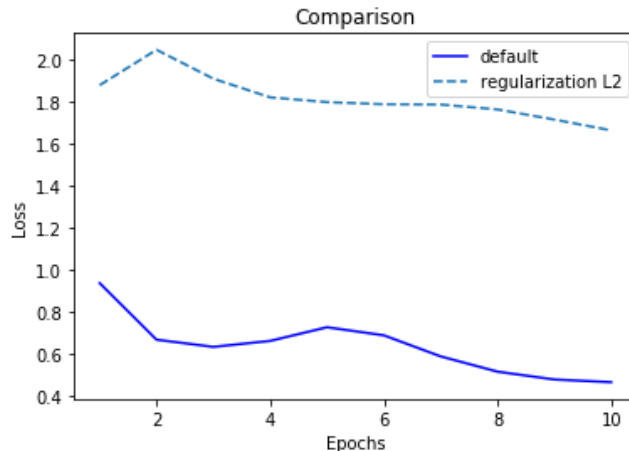


Figura 6. 3: Loss-ul după regularizarea L1, comparativ cu cel inițial

## 6.2 CNN2

Următorul experiment diferă de primul prin faptul că numărul de clase este acum 10 (am antrenat, deci, modelul, pentru a recunoaște 10 vorbitori). Restul parametrilor menționați rămân aceiași.

În acest caz, acuratețea este semnificativ afectată, fiind de 40.72%. Matricea de confuzie, precizia și recall-ul sunt, de asemenea, scăzute, după cum se observă din figura 6.4:

```

precision    recall  f1-score   support

0           0.00      0.00      0.00        16
1           0.48      0.89      0.63        18
2           0.61      0.51      0.56        43
3           0.00      0.00      0.00        20
4           0.00      0.00      0.00        13
5           0.65      0.43      0.52        30
6           0.27      1.00      0.43        14
7           0.25      0.81      0.39        16
8           0.50      0.08      0.14        12
9           0.00      0.00      0.00        12

accuracy          0.41        194
macro avg         0.28      0.37      0.27        194
weighted avg      0.35      0.41      0.33        194

Confusion matrix:
[[ 0  0  0  1  0  6  0  9  0  0]
 [ 0 16  2  0  0  0  0  0  0  0]
 [ 0 16 22  0  0  0  1  4  0  0]
 [ 0  0  2  0  0  1  7 10  0  0]
 [ 0  0  5  0  0  0  5  2  1  0]
 [ 0  1  2  0  0 13  8  6  0  0]
 [ 0  0  0  0  0  0 14  0  0  0]
 [ 0  0  0  0  0  0  3 13  0  0]
 [ 0  0  2  0  0  0  4  5  1  0]
 [ 0  0  1  0  0  0  9  2  0  0]]

```

Figura 6. 4: Valorile metricilor pentru CNN2

Nici în acest caz regularizarea nu aduce îmbunătățiri din punct de vedere al loss-ului.

### 6.3 CNN3

În următorul experiment, am scăzut batch size-ul la 11, pe principiul că, cu cât avem mai multe batch-uri mai mici, cu atât se actualizează și se îmbunătățesc mai des weight-urile modelului. Am folosit primii doi vorbitori, restul parametrilor menționați rămânând aceiași.

Am obținut o acuratețe la testare de 100%, această valoare fiind valabilă și pentru precizie și recall. Matricea de confuzie este redată în figura 6.5:

```

Confusion matrix:
[[13  0]
 [ 0 29]]

```

Figura 6. 5: Matricea de confuzie pentru CNN3

Observând această îmbunătățire pentru doi vorbitori, am decis să realizez un experiment similar, cu batch size-ul redus, pentru 10 vorbitori:

## 6.4 CNN4

După cum am menționat mai sus, în acest experiment am redus batch size-ul și am antrenat rețeaua pentru 10 vorbitori, dimensiunile imaginilor extrase fiind aceleași (45x2900). Am obținut o acuratețe la testare de 71.13%. Precizia, recall-ul și matricea de confuzie arată ca în figura 6.6:

```
In [10]: evaluate.display_metrics(model, 'CNN4', result_sets)
```

	precision	recall	f1-score	support
0	0.65	0.81	0.72	16
1	0.71	0.94	0.81	18
2	0.97	0.84	0.90	43
3	0.41	0.60	0.49	20
4	1.00	0.15	0.27	13
5	0.92	0.73	0.81	30
6	0.92	0.79	0.85	14
7	0.64	0.88	0.74	16
8	1.00	0.08	0.15	12
9	0.43	0.83	0.57	12
accuracy			0.71	194
macro avg	0.76	0.67	0.63	194
weighted avg	0.79	0.71	0.70	194

Confusion matrix:

```
[[13  0  1  0  0  0  0  2  0  0]
 [ 0 17  0  1  0  0  0  0  0  0]
 [ 0  5 36  1  0  0  0  0  0  1]
 [ 3  0  0 12  0  0  0  1  0  4]
 [ 1  1  0  5  2  0  0  0  0  4]
 [ 0  0  0  4  0 22  0  4  0  0]
 [ 0  0  0  1  0  2 11  0  0  0]
 [ 1  0  0  0  0  0  1 14  0  0]
 [ 2  0  0  4  0  0  0  1  1  4]
 [ 0  1  0  1  0  0  0  0  0 10]]
```

Figura 6. 6: Valorile metricilor pentru CNN4

Studiind graficele din figura 6.7 am observat faptul că modelul ar mai putea fi antrenat un număr de epoci, deoarece lossul este în scădere și acuratețea este în creștere (acestea nu au ajuns la convergență):

```
In [11]: evaluate.plot_history(history)
```

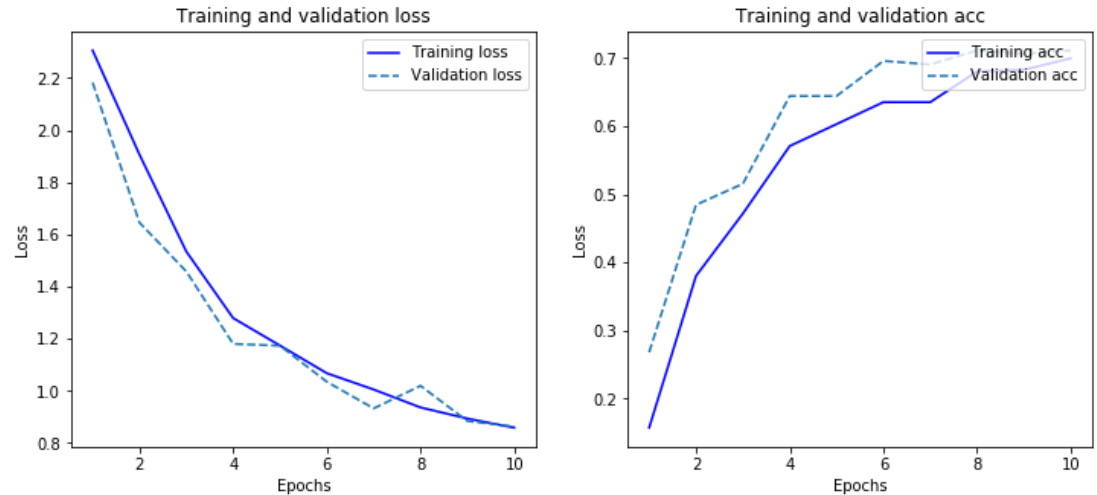


Figura 6. 7: Evoluția acurateții și a loss-ului pentru CNN4, antrenat pe 10 epoci

## 6.5 CNNCw

În următorul experiment am adus mai multe îmbunătățiri, și anume:

- am antrenat modelul timp de 100 de epoci
- am luat din fiecare clip câte 3 secunde, astfel nu am mai introdus acel padding în imagini, care a dus cel mai probabil la erori de clasificare (a fost perceput ca fiind parte din feature).
- am folosit class weights, deoarece am observat că setul de date nu este echilibrat; pentru unii vorbitori avem mai multe exemple decât pentru ceilalți (uneori aproape dublu).

Rezultatele s-au îmbunătățit semnificativ, acuratețea la testare fiind acum de 91.23%, iar precizia, recall-ul și matricea de confuzie fiind ilustrate în figura 6.8:

	precision	recall	f1-score	support
0	0.84	1.00	0.91	16
1	1.00	1.00	1.00	18
2	1.00	1.00	1.00	43
3	0.94	0.85	0.89	20
4	1.00	0.54	0.70	13
5	0.84	0.90	0.87	30
6	0.88	1.00	0.93	14
7	1.00	0.88	0.93	16
8	0.83	0.83	0.83	12
9	0.73	0.92	0.81	12
accuracy			0.91	194
macro avg	0.91	0.89	0.89	194
weighted avg	0.92	0.91	0.91	194

Confusion matrix:

```
[[16  0  0  0  0  0  0  0  0  0]
 [ 0 18  0  0  0  0  0  0  0  0]
 [ 0  0 43  0  0  0  0  0  0  0]
 [ 1  0  0 17  0  2  0  0  0  0]
 [ 2  0  0  0  7  0  1  0  0  3]
 [ 0  0  0  0  0 27  1  0  2  0]
 [ 0  0  0  0  0  0 14  0  0  0]
 [ 0  0  0  1  0  1  0 14  0  0]
 [ 0  0  0  0  0  1  0  0 10  1]
 [ 0  0  0  0  0  1  0  0  0 11]]
```

Figura 6. 8: Valorile metricilor pentru CNNCw

## 6.6 CNNCwHTK

În acest experiment am schimbat modul de extragere a mfcc-urilor, de la normal (htk = False) la HTK (htk = True). În rest, am păstrat parametrii rețelei la fel. Rezultatele nu au fost mai bune, am obținut o acuratețe la testare de 88.65% și valorile metricilor din figura 6.9.

	precision	recall	f1-score	support
0	1.00	0.94	0.97	16
1	0.85	0.94	0.89	18
2	0.98	0.98	0.98	43
3	0.86	0.95	0.90	20
4	0.89	0.62	0.73	13
5	0.82	0.90	0.86	30
6	1.00	0.86	0.92	14
7	0.88	0.94	0.91	16
8	0.80	0.67	0.73	12
9	0.69	0.75	0.72	12
accuracy			0.89	194
macro avg	0.88	0.85	0.86	194
weighted avg	0.89	0.89	0.88	194

Confusion matrix:

```
[[15  0  1  0  0  0  0  0  0  0]
 [ 0 17  0  1  0  0  0  0  0  0]
 [ 0  1 42  0  0  0  0  0  0  0]
 [ 0  0  0 19  0  0  0  1  0  0]
 [ 0  2  0  0  8  1  0  0  0  2]
 [ 0  0  0  1  0 27  0  0  2  0]
 [ 0  0  0  0  0  2 12  0  0  0]
 [ 0  0  0  0  0  1  0 15  0  0]
 [ 0  0  0  0  1  1  0  0  8  2]
 [ 0  0  0  1  0  1  0  1  0  9]]
```

Figura 6. 9: CNNCwHTK

## 6.7 CNNCwSr

Am schimbat din nou modul de extragere a mfcc-urilor, mărind rata de eșantionare (sampling rate) de la 22050, cât era implicit, la 30050. Am luat câte 5 secunde din fiecare clip, iar dimensiunea imaginilor a devenit 45x300. Am observat o ușoară îmbunătățire în valorile metricilor: acuratețea la testare a devenit 93.81%, restul metricilor fiind cele din figura 6.10.

Cele mai mari îmbunătățiri la precizie și recall se observă din figură pentru clasele 0, 4 și 8, deci observăm că chiar dacă în acuratețe modelul nu a crescut foarte mult, am avut de câștigat din aplicarea acestei tehnici.



```

precision    recall  f1-score   support

0           1.00      1.00      1.00        16
1           1.00      1.00      1.00        18
2           1.00      1.00      1.00        43
3           0.90      0.90      0.90        20
4           0.92      0.92      0.92        13
5           0.90      0.87      0.88        30
6           0.88      1.00      0.93        14
7           1.00      0.88      0.93        16
8           0.86      1.00      0.92        12
9           0.82      0.75      0.78        12

accuracy          0.94      194
macro avg         0.93      194
weighted avg      0.94      194

Confusion matrix:
[[16  0  0  0  0  0  0  0  0  0]
 [ 0 18  0  0  0  0  0  0  0  0]
 [ 0  0 43  0  0  0  0  0  0  0]
 [ 0  0  0 18  0  1  0  0  0  1]
 [ 0  0  0  0 12  0  0  0  0  1]
 [ 0  0  0  1  0 26  1  0  2  0]
 [ 0  0  0  0  0  0 14  0  0  0]
 [ 0  0  0  0  0  1  1 14  0  0]
 [ 0  0  0  0  0  0  0  0 12  0]
 [ 0  0  0  1  1  1  0  0  0  9]]

```

Figura 6. 10: CNNCwSr

## 6.8 CNNMF

În acest experiment am folosit o abordare similară celei de la CNNCw (mai puțin partea cu class weights). Am clasificat exemple pentru doi vorbitori, un bărbat și o femeie, A.J. Buckley și Abbie Cornish. Rezultatele obținute sunt cele din figura 6.11:

```

In [12]: evaluate.display_metrics(model, 'CNNMF', result_sets)

precision    recall  f1-score   support

0           0.87      0.93      0.90        14
1           0.96      0.93      0.94        27

accuracy          0.93      41
macro avg         0.91      41
weighted avg      0.93      41

Confusion matrix:
[[13  1]
 [ 2 25]]

```

Figura 6. 11: CNNMF

Acuratețea la testare a acestui model este de 92.68%.

## 6.9 CNNMFDiff

Acest experiment este similar cu cel dinainte, doar că vorbitorii aleși de această dată sunt un bărbat cu voce groasă și o femeie cu voce subțire, pentru a verifica faptul că clasificarea va fi mai ușoară pentru vorbitori ale căror voci diferă semnificativ. Ipoteza a fost confirmată, după cum se vede din figura 6.12:

```
In [8]: evaluate.evaluate_model(model, 'CNNMFDiff', result_sets)

Training Accuracy: 1.0
Testing Accuracy: 1.0

In [9]: evaluate.display_metrics(model, 'CNNMFDiff', result_sets)
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	11
1	1.00	1.00	1.00	36
accuracy			1.00	47
macro avg	1.00	1.00	1.00	47
weighted avg	1.00	1.00	1.00	47

```
Confusion matrix:
[[11  0]
 [ 0 36]]
```

Figura 6. 12: CNNMFDiff

## 6.10 CNN2Men

Am decis să realizez un experiment similar pentru doi bărbați cu voci asemănătoare (ambii tenori). După cum mă așteptam, rezultatele au avut de suferit ușor (fig. 6.13).

```
In [8]: evaluate.evaluate_model(model, 'CNN2Men', result_sets)

Training Accuracy: 0.8986486
Testing Accuracy: 0.8378378

In [9]: evaluate.display_metrics(model, 'CNN2Men', result_sets)
```

	precision	recall	f1-score	support
0	0.84	0.84	0.84	19
1	0.83	0.83	0.83	18
accuracy			0.84	37
macro avg	0.84	0.84	0.84	37
weighted avg	0.84	0.84	0.84	37

```
Confusion matrix:
[[16  3]
 [ 3 15]]
```

Figura 6. 13: CNN2Men

### 6.11 CNNTwoInputs

Am antrenat timp de 100 de epoci un model de rețea convoluțională cu două tipuri de intrări, mfcc-uri și melspectrograme. Dimensiunile mfcc-urilor au fost de 45x130 (ca la CNNCw), iar ale melspectrogramelor de 32x110. Am folosit 10 vorbitori și class weights. Acuratețea a rămas aceeași ca la CNNCw, dar s-au observat îmbunătățiri semnificative la precizie și recall pentru clasele 4 și 9 (fig. 6.14):

	precision	recall	f1-score	support
0	0.94	1.00	0.97	16
1	0.86	1.00	0.92	18
2	1.00	0.95	0.98	43
3	0.89	0.80	0.84	20
4	0.85	0.85	0.85	13
5	0.93	0.83	0.88	30
6	0.87	0.93	0.90	14
7	1.00	1.00	1.00	16
8	0.71	0.83	0.77	12
9	0.92	0.92	0.92	12
accuracy			0.91	194
macro avg	0.90	0.91	0.90	194
weighted avg	0.92	0.91	0.91	194

```

Confusion matrix:
[[16  0  0  0  0  0  0  0  0  0]
 [ 0 18  0  0  0  0  0  0  0  0]
 [ 0  2 41  0  0  0  0  0  0  0]
 [ 1  0  0 16  0  1  0  0  2  0]
 [ 0  0  0  0 11  0  1  0  0  1]
 [ 0  0  0  1  1 25  1  0  2  0]
 [ 0  0  0  0  0  1 13  0  0  0]
 [ 0  0  0  0  0  0  0 16  0  0]
 [ 0  1  0  1  0  0  0  0 10  0]
 [ 0  0  0  0  1  0  0  0  0 11]]

```

Figura 6. 14: CNNTwoInputs

## 6.12 MobileNet1

În acest experiment am folosit rețeaua pre-antrenată MobileNetV2, descrisă în capitolul 4, și am lăsat să se antreneze 35 de layere din cele 155. A fost nevoie, pentru a folosi MobileNet, să transform imaginile mfcc din grayscale (1 canal) în RGB (3 canale), repetând aceleași valori pe toate cele 3 canale. Am adăugat peste MobileNet un layer de tip global average pooling și un layer dens cu funcția de activare softmax, pentru clasificare. Am făcut clasificare binară, între primii doi vorbitori din dataset-ul VoxCeleb. Dimensiunea aleasă pentru imagini a fost de 128x128, pentru că MobileNet dă rezultate mai bune pe imagini pătrate (a fost antrenată pe astfel de imagini). Am antrenat rețeaua obținută timp de 100 de epoci, cu un batch size de 13 și cu o rată de învățare de 0.0001. Rezultatele obținute, în afară de acuratețea la testare de 88.09%, sunt cele din figura 6.15:

```

                precision    recall  f1-score   support

     0           0.83         0.77         0.80         13
     1           0.90         0.93         0.92         29

 accuracy          0.88         0.88         0.88         42
 macro avg         0.87         0.85         0.86         42
 weighted avg      0.88         0.88         0.88         42

Confusion matrix:
[[10  3]
 [ 2 27]]

```

Figura 6. 15: MobileNet1

### 6.13 MobileNet2

În mod similar, am antrenat un model cu o arhitectură identică, pe 10 vorbitori. Faptul că rezultatele au fost oarecum dezamăgitoare arată că această soluție nu este scalabilă, de aceea nu a fost luată în considerare pentru experimente în continuare; rezultatele sunt: acuratețe la testare de 51.54%, și valorile metricilor conform figurii 6.16:

```

                precision    recall  f1-score   support

     0           0.50         0.25         0.33         16
     1           0.65         0.72         0.68         18
     2           0.52         0.77         0.62         43
     3           0.50         0.35         0.41         20
     4           1.00         0.08         0.14         13
     5           0.79         0.37         0.50         30
     6           0.59         0.71         0.65         14
     7           0.48         0.69         0.56         16
     8           0.39         0.58         0.47         12
     9           0.19         0.25         0.21         12

 accuracy          0.52         0.52         0.52        194
 macro avg         0.56         0.48         0.46        194
 weighted avg      0.58         0.52         0.49        194

Confusion matrix:
[[ 4  1  6  1  0  0  0  0  3  1]
 [ 0 13  3  1  0  0  0  0  0  1]
 [ 1  3 33  0  0  0  0  2  0  4]
 [ 1  0  5  7  0  0  0  2  3  2]
 [ 0  2  0  1  1  0  1  2  3  3]
 [ 0  0 10  1  0 11  4  3  0  1]
 [ 0  0  0  0  0  2 10  2  0  0]
 [ 0  0  2  0  0  1  2 11  0  0]
 [ 1  0  2  0  0  0  0  1  7  1]
 [ 1  1  2  3  0  0  0  0  2  3]]

```

Figura 6. 16: MobileNet 2

### 6.14 DenseNet1

O altă arhitectură cu rețea pre-antrenată, în schimb, s-a dovedit a da rezultate promițătoare, și anume cea cu DenseNet121. Am lăsat să se antreneze layerurile de la 300 în sus (până la 428), imaginile de antrenament și de test au fost tot mfcc-uri de 128x128, RGB, batch size-ul, numărul de epoci și rata de învățare au fost identice cu cele din experimentul anterior. Am obținut 82.98% acuratețe la validare, și valorile metricilor din figura 6.17:

```

precision    recall  f1-score   support

0           0.78        0.88        0.82         16
1           0.82        1.00        0.90         18
2           0.97        0.88        0.93         43
3           0.64        0.80        0.71         20
4           0.64        0.54        0.58         13
5           0.94        0.97        0.95         30
6           0.71        0.86        0.77         14
7           1.00        0.94        0.97         16
8           1.00        0.58        0.74         12
9           0.56        0.42        0.48         12

accuracy                    0.83        194
macro avg                   0.80        0.79        0.79        194
weighted avg                0.84        0.83        0.83        194

Confusion matrix:
[[14  0  0  1  0  0  0  0  0  1]
 [ 0 18  0  0  0  0  0  0  0  0]
 [ 1  1 38  1  1  0  1  0  0  0]
 [ 0  1  0 16  2  0  1  0  0  0]
 [ 2  0  0  1  7  0  1  0  0  2]
 [ 0  0  0  0  0 29  1  0  0  0]
 [ 0  0  0  1  1  0 12  0  0  0]
 [ 0  0  0  0  0  0  1 15  0  0]
 [ 0  1  0  2  0  1  0  0  7  1]
 [ 1  1  1  3  0  1  0  0  0  5]]

```

Figura 6. 17: DenseNet1

### 6.15 DenseNetMelSpec

Am realizat și un experiment cu aceeași arhitectură, dar imagini de intrare melspectrograme cu 3 canale, de dimensiuni 32x110. Am obținut 52.06% acuratețe la validare și valorile metricilor din figura 6.18:

```

0      0.40      0.25      0.31      16
1      0.50      0.78      0.61      18
2      0.62      0.56      0.59      43
3      0.75      0.30      0.43      20
4      0.23      0.46      0.31      13
5      0.68      0.77      0.72      30
6      0.59      0.71      0.65      14
7      0.37      0.44      0.40      16
8      0.40      0.17      0.24      12
9      0.62      0.42      0.50      12

accuracy      0.52      0.48      0.52      194
macro avg     0.52      0.48      0.47      194
weighted avg  0.55      0.52      0.51      194

Confusion matrix:
[[ 4  0  3  1  5  1  0  0  1  1]
 [ 1 14  3  0  0  0  0  0  0  0]
 [ 2  8 24  0  4  1  0  4  0  0]
 [ 1  2  5  6  4  1  0  1  0  0]
 [ 0  2  2  0  6  0  1  0  1  1]
 [ 0  0  0  0  1 23  2  4  0  0]
 [ 0  0  0  1  0  1 10  2  0  0]
 [ 0  1  0  0  1  5  2  7  0  0]
 [ 2  0  2  0  3  2  0  0  2  1]
 [ 0  1  0  0  2  0  2  1  1  5]]

```

Figura 6. 18: DenseNetMelSpec

Studiind imaginile, am suspectat faptul că abordarea bazată pe melspectrograme nu are la fel de mult succes pentru că aceste imagini nu sunt la fel de complexe ca și mfcc-urile, nu cuprind la fel de multă și de variată informație în ele. Pentru a demonstra și a susține matematic acest lucru, am calculat entropia medie a 10 mfcc-uri și a 10 melspectrograme, extrase din aceleași clipuri, și am obținut următorul rezultat: entropia medie a mfcc-urilor este de 11.26, iar entropia medie a melspectrogramelor este de 6.74. Pe lângă acest rezultat, avem și histogramele realizate pe mfcc-ul, respective melspectrograma extrase din același clip, care susțin ipoteza că mfcc-urile sunt mai complexe. Cele două histograme sunt înfățișate în figurile 6.19 și 6.20:

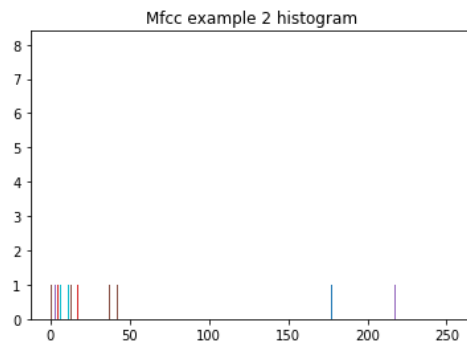


Figura 6. 19: Histogramă mfcc

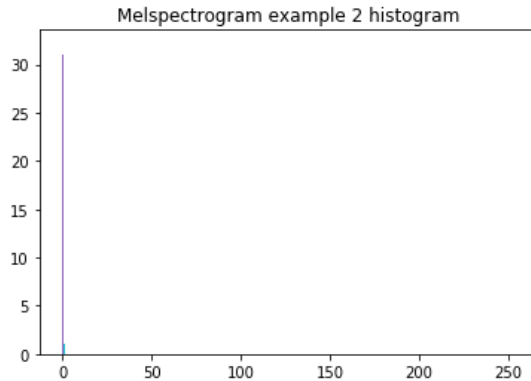


Figura 6. 20: Histogramă melspectrogram

### 6.16 DenseNet2Men

În continuare, am mers pe abordarea cu mfcc, și am construit un model cu DenseNet, care distinge între cei doi bărbați cu voci de tenor menționați într-un experiment anterior. În afară de faptul că avem doar doi vorbitori și nu folosim class weights, experimentul are aceiași parametri ca și DenseNet1. Am obținut 94.59% acuratețe la testare, și valori ale celorlalte metrice conform figurii 6.21:

	precision	recall	f1-score	support
0	1.00	0.89	0.94	19
1	0.90	1.00	0.95	18
accuracy			0.95	37
macro avg	0.95	0.95	0.95	37
weighted avg	0.95	0.95	0.95	37

Confusion matrix:

```
[[17  2]
 [ 0 18]]
```

Figura 6. 21: DenseNet2Men

### 6.17 DenseNetMF

Asemănător cu exemplul de mai sus, am antrenat un model și pentru cei doi vorbitori din experimentul CNNMF, A.J. Buckley și Abbie Cornish. Am obținut 100% acuratețe la validare și 100% precizie și recall pentru ambele clase, după cum se poate vedea și în figura 6.22:



```

              precision    recall  f1-score   support

     0         1.00      1.00      1.00         14
     1         1.00      1.00      1.00         27

 accuracy          1.00          1.00          1.00         41
 macro avg          1.00          1.00          1.00         41
 weighted avg          1.00          1.00          1.00         41

Confusion matrix:
[[14  0]
 [ 0 27]]

```

Figura 6. 22: DenseNetMF

### 6.18 DenseNetMFDiff

Tot în același mod, am antrenat un model care să distingă între un bărbat cu voce groasă și o femeie cu voce subțire, aceiași din experimentul similar cu CNN. Rezultatele sunt tot de 100% acuratețe, precizie și recall pentru ambele clase.

### 6.19 Verificarea pe date noi

Am verificat faptul că proiectul funcționează în momentul în care furnizăm înregistrări noi pentru a fi recunoscute. Am apelat funcția `print_prediction` pentru mai multe clipuri, pentru fiecare vorbitor, clipuri luate din 1-2 interviuri cu acesta. Pentru a mă asigura că nu verific cu clipuri care au fost date pentru antrenament, am căutat pe cât posibil interviuri făcute după anul creării setului de date VoxCeleb, și anume 2017.

Clipurile au fost tăiate printr-o metodă automată, folosind biblioteca `pydub`, în notebook-ul `clips`.

Rezultatele inițiale ale acestei verificări sunt regăsite în notebook-ul `TestNotebookCNNCw`. Am testat unul dintre cele mai performante modele pe care le-am obținut, `CNNCw`, cel antrenat pe 10 vorbitori timp de 100 de epoci. Pentru 6 din acești vorbitori, și anume Aamir Khan, Abbie Cornish, Abigail Breslin, A. R. Rahman, Adam Beach și Aaron Yoo, sistemul a recunoscut clipurile într-o proporție destul de bună (peste jumătate, iar la unii vorbitori toate clipurile). Pentru Abigail Spencer, sistemul a recunoscut unele dintre clipuri, dar sub jumătate, nu la fel de bine ca și pentru cei 6 menționați. Pentru ultimii 3 vorbitori, Adam Brody, A.J. Buckley și Aaron Tveit, rezultatele nu au fost foarte bune, sistemul recunoscând sub 3 clipuri din 10.

Pentru a descoperi care este cauza acestui lucru, am ales să descarc două noi interviuri cu vorbitorul care părea a fi cel mai puțin recunoscut, Aaron Tveit, să adaug în setul de date clipurile extrase din aceste două interviuri și să antrenez un nou model pe acest nou set de date. Diferența a fost majoră, de această dată sistemul a recunoscut clipurile ca aparținând vorbitorului corect în proporție de 7/12 (7 din 12 clipuri). Studiind probabilitățile obținute, am observat și faptul că pentru clipurile clasificate incorect vorbitorul prezis imediat următor este cel corect. Un exemplu de astfel de clasificare este cea din figura 6.23:

```
for i in range(1, 12):
    print_prediction('./test_clips/aaron_tveit/1/test_aaron_tveit' + str(i))
```

The predicted class is: Aamir\_Khan

A.J._Buckley	:	0.00000002392941844675533502595499
A.R._Rahman	:	0.00469308439642190933227539062500
Aamir_Khan	:	0.69711554050445556640625000000000
Aaron_Tveit	:	0.29615387320518493652343750000000
Aaron_Yoo	:	0.00178828788921236991882324218750
Abbie_Cornish	:	0.00000654403947919490747153759003
Abigail_Breslin	:	0.00000000019912199666904939476808
Abigail_Spencer	:	0.00018991390243172645568847656250
Adam_Beach	:	0.00000100963461591163650155067444
Adam_Brody	:	0.00005170537406229414045810699463

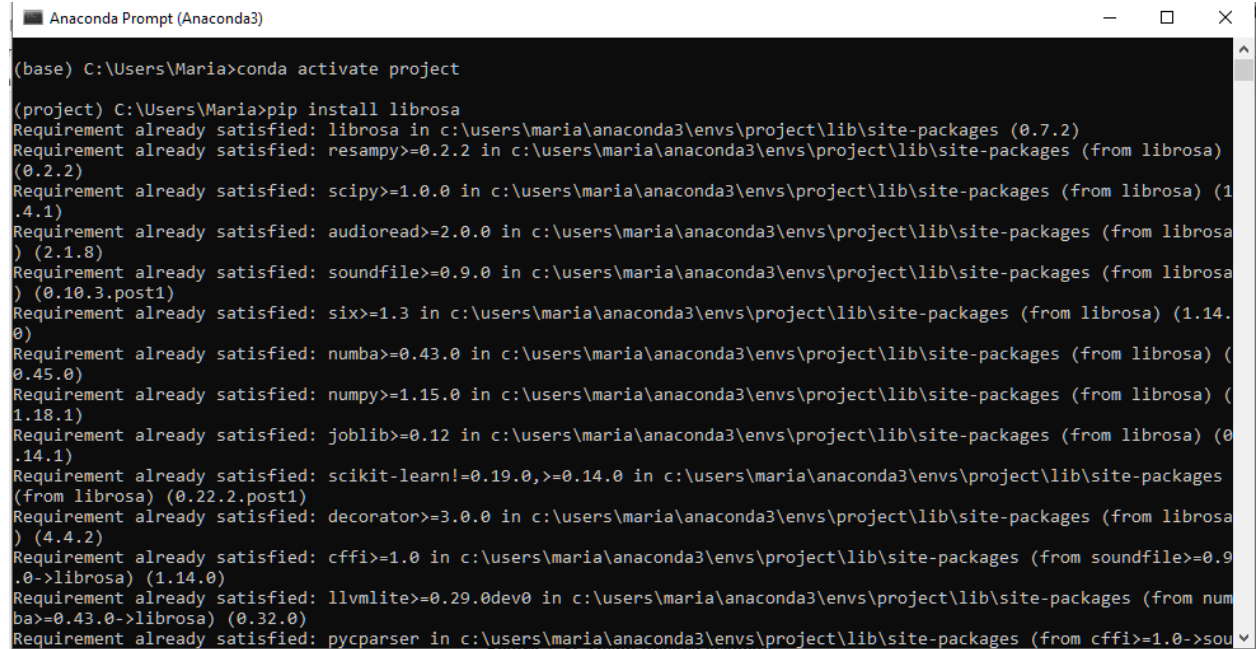
Rezultatele pentru toți vorbitorii sunt în notebook-ul de test TestNotebookCNNImproved.

Concluzia trasă în urma testării pe date noi este că setul de date VoxCeleb nu este suficient de larg pentru unii vorbitori (nu este suficient de reprezentativ pentru caracteristicile vocilor lor), dar dacă acest set de date este lărgit se pot obține rezultate foarte bune.

## Capitolul 7. Manual de instalare și utilizare

Pentru a utiliza modelele create, nu există deocamdată o interfață grafică care să facă mai ușor procesul recunoașterii de vorbitor, dar modele se pot utiliza din jupyter notebook-uri asemănătoare celor de test (TestNotebookCNNCw, de exemplu).

Utilizatorul trebuie să aibă instalat Python 3.7, Jupyter și bibliotecile de Python necesare execuției funcțiilor din proiect (tensorflow, tensorflow.keras, pandas, numpy, librosa, matplotlib, și tot restul bibliotecilor importate în modulele proiectului). Lista acestor biblioteci este regăsită în fișierul libraries.txt. Pentru a le instala se poate folosi o linie de comandă de tipul celei din anaconda, ca în figura 7.1:



```

Anaconda Prompt (Anaconda3)

(base) C:\Users\Maria>conda activate project

(project) C:\Users\Maria>pip install librosa
Requirement already satisfied: librosa in c:\users\maria\anaconda3\envs\project\lib\site-packages (0.7.2)
Requirement already satisfied: resampy>=0.2.2 in c:\users\maria\anaconda3\envs\project\lib\site-packages (from librosa) (0.2.2)
Requirement already satisfied: scipy>=1.0.0 in c:\users\maria\anaconda3\envs\project\lib\site-packages (from librosa) (1.4.1)
Requirement already satisfied: audioread>=2.0.0 in c:\users\maria\anaconda3\envs\project\lib\site-packages (from librosa) (2.1.8)
Requirement already satisfied: soundfile>=0.9.0 in c:\users\maria\anaconda3\envs\project\lib\site-packages (from librosa) (0.10.3.post1)
Requirement already satisfied: six>=1.3 in c:\users\maria\anaconda3\envs\project\lib\site-packages (from librosa) (1.14.0)
Requirement already satisfied: numba>=0.43.0 in c:\users\maria\anaconda3\envs\project\lib\site-packages (from librosa) (0.45.0)
Requirement already satisfied: numpy>=1.15.0 in c:\users\maria\anaconda3\envs\project\lib\site-packages (from librosa) (1.18.1)
Requirement already satisfied: joblib>=0.12 in c:\users\maria\anaconda3\envs\project\lib\site-packages (from librosa) (0.14.1)
Requirement already satisfied: scikit-learn!=0.19.0,>=0.14.0 in c:\users\maria\anaconda3\envs\project\lib\site-packages (from librosa) (0.22.2.post1)
Requirement already satisfied: decorator>=3.0.0 in c:\users\maria\anaconda3\envs\project\lib\site-packages (from librosa) (4.4.2)
Requirement already satisfied: cffi>=1.0 in c:\users\maria\anaconda3\envs\project\lib\site-packages (from soundfile>=0.9.0->librosa) (1.14.0)
Requirement already satisfied: llvmlite>=0.29.0dev0 in c:\users\maria\anaconda3\envs\project\lib\site-packages (from numba>=0.43.0->librosa) (0.32.0)
Requirement already satisfied: pycparser in c:\users\maria\anaconda3\envs\project\lib\site-packages (from cffi>=1.0->soundfile->librosa) (2.10)
  
```

Figura 7. 1: Instalarea bibliotecii librosa

Trebuie avut grijă ca bibliotecile să se instaleze în același environment ca și cel din care vom rula proiectul (cel din care pornim jupyter notebook). În cazul de mai sus, environment-ul se numește project, a fost creat cu comanda “conda create - - name project”, și a fost activate cu comanda “conda activate project”.

Pentru a utiliza un model, utilizatorul trebuie să copieze notebook-ul TestNotebookCNNCw și să îl modifice după cum dorește, să scrie numele modelului dorit în funcția load\_weights, pentru a-l încărca (în acest tip de notebook poate încărca orice model mai puțin cel cu două input-uri). Pentru a folosi modelele pre-antrenate este nevoie să decommenteze liniile comentate, cele care în figura 7.2 apar cu chenar roșu, și să comenteze liniile care sunt în chenarul verde:

```

In [3]: model = learn.build_CNN_model(num_rows, num_columns, num_labels)
        #base_model = learn.get_densenet(num_rows, num_columns, num_channels, num_labels, fine_tune_at)
        #model = learn.build_model_densenet(base_model, num_labels)

In [4]: model.load_weights("saved_models/weights.best.CNNCwImproved.hdf5")
        #model.load_weights("saved_models/weights.best.DenseNet1.hdf5")
  
```

Figura 7. 2: Liniile care trebuie de-comentate și cele care trebuie comentate

În continuare, trebuie să creeze o celulă goală, în care să apeleze funcția `evaluate.print_prediction`, unde primul parametru să fie calea spre fișierul `.wav` ce se dorește a fi analizat, ca în figura 7.3:

```
: evaluate.print_prediction('./exampleFile.wav', model, num_rows, num_columns, num_channels, num_seconds)

: for i in range(4, 18):
  evaluate.print_prediction('./test_clips/abbie_cornish/2/test_abbie_cornish' + str(i) + '.wav', model, num_rows, num_columns,
```

Figura 7. 3: Apelul funcției `evaluate`

În final, este necesar ca utilizatorul să ruleze notebook-ul celulă cu celulă, până la celula nou introdusă, inclusiv. Rezultatele oferite de model vor apărea în output-ul celulei noi, sub forma unui verdict de tipul “The predicted class is:”, iar sub el vor fi probabilitățile pentru fiecare clasă.

## Capitolul 8. Concluzii

Am dezvoltat un sistem care recunoaște vorbitori din înregistrări, doar dintre cei care fac parte din setul de date VoxCeleb. Acuratețea maximă a modelelor obținute pe acest set de date, pentru 10 vorbitori, a fost de 93% (cel mai bun model fiind CNNTwoInputs). De asemenea, am demonstrat faptul că acest set de date nu este întocmai reprezentativ pentru vocile vorbitorilor, și că dacă este extins, rețeaua nou antrenată va da rezultate mai bune pe date noi. Concluzia la care am ajuns este că vocea umană își schimbă caracteristicile odată cu înaintarea în vârstă (în mod special frecvența, care este caracteristica principală extrasă în spectrograme), și de aceea setul de date trebuie actualizat periodic și rețeaua re-antrenată pentru a reflecta aceste schimbări. Mai mult decât atât, anumite voci variază în tonalitate mai mult decât altele, și anume vocile cântăreților<sup>[13]</sup>, de aceea pentru acest tip de vorbitori este nevoie de mai multe date de antrenament. Acesta a fost cazul lui Aaron Tveit, care este și cântăreț.

Antrenând și modele cu rețele pre-antrenate, am observat că valorile acurateței, ale preciziei și recall-ului sunt un pic mai bune pentru 2 vorbitori, dar mai rele pentru 10 vorbitori, ceea ce ar putea fi din cauză că acele rețele au fost pre-antrenate pe imagini care nu seamănă cu mfcc-urile și cu melspectrogramele.

Ca și îmbunătățiri, s-ar putea lărgi setul de date pentru vorbitorii deja existenți, sau s-ar putea adăuga noi vorbitori, pentru a putea detecta cât mai multe persoane. O altă îmbunătățire posibilă ar fi crearea unei interfețe grafice pentru ca aplicația să fie utilizată mult mai ușor de către un utilizator fără cunoștințe tehnice. De asemenea, s-ar putea încerca continuarea experimentelor, modificarea unor anumiți parametri ai rețelei (learning rate, de exemplu), folosirea altor rețele pre-antrenate sau folosirea unei RNN, cu sunetul tratat sub formă de serie temporală.

## Bibliografie

- [1] Andre Ye, 2020, “A Guide to Neural Network Layers with Applications in Keras”, 2020  
(explicații legate de layere), <https://towardsdatascience.com/a-guide-to-neural-network-layers-with-applications-in-keras-40ccb7ebb57a>
- [2] Kishore Prahallad, curs din cadrul universității Carnegie Mellon (despre mfcc și melspectrograme), [http://www.speech.cs.cmu.edu/15-492/slides/03\\_mfcc.pdf](http://www.speech.cs.cmu.edu/15-492/slides/03_mfcc.pdf)  
<https://towardsdatascience.com/audio-classification-using-fastai-and-on-the-fly-frequency-transforms-4dbe1b540f89>
- [3] Exemplu de model cu două intrări și două ieșiri, tutorial tensorflow, [https://www.tensorflow.org/guide/keras/train\\_and\\_evaluate#passing\\_data\\_to\\_multi-input\\_multi-output\\_models](https://www.tensorflow.org/guide/keras/train_and_evaluate#passing_data_to_multi-input_multi-output_models)
- [4] Mike Smales, exemplu de clasificare de sunete cu rețea convoluțională, <https://medium.com/@mikesmales/sound-classification-using-deep-learning-8bc2aa1990b7>
- [5] Nagesh Singh Chauhan, “Audio Data Analysis Using Deep Learning With Python (Part 1)” (exemplu librosa): <https://www.kdnuggets.com/2020/02/audio-data-analysis-deep-learning-python-part-1.html>
- [6] Ahmed F. Gad, “Part 3: Image Classification using Features Extracted by Transfer Learning in Keras”, 2019 (exemplu de model Sequential cu rețea pre-antrenată): [https://www.alibabacloud.com/blog/part-3-image-classification-using-features-extracted-by-transfer-learning-in-keras\\_595291](https://www.alibabacloud.com/blog/part-3-image-classification-using-features-extracted-by-transfer-learning-in-keras_595291)
- [7] Exemplu cu MobileNet, [https://www.tensorflow.org/tutorials/images/transfer\\_learning#create\\_the\\_base\\_model\\_from\\_the\\_pre-trained\\_convnets](https://www.tensorflow.org/tutorials/images/transfer_learning#create_the_base_model_from_the_pre-trained_convnets)  
[https://developer.ridgerun.com/wiki/index.php?title=Keras\\_with\\_MobilenetV2\\_for\\_Deep\\_Learning#Training\\_the\\_model](https://developer.ridgerun.com/wiki/index.php?title=Keras_with_MobilenetV2_for_Deep_Learning#Training_the_model)
- [8] Gao Huang, Zhuang Liu, Geoff Pleiss, Laurens van der Maaten și Kilian Q. Weinberger, “Convolutional Networks with Dense Connectivity”, în *Ieee Transactions On Pattern Analysis And Machine Intelligence*, 2019, [http://www.gaohuang.net/papers/DenseNet\\_Journal.pdf](http://www.gaohuang.net/papers/DenseNet_Journal.pdf)
- [9] Tutorial TensorBoard: [https://www.tensorflow.org/guide/keras/train\\_and\\_evaluate#visualizing\\_loss\\_and\\_metrics\\_during\\_training](https://www.tensorflow.org/guide/keras/train_and_evaluate#visualizing_loss_and_metrics_during_training)
- [10] Aurélien Géron, “Hands-On Machine Learning with Scikit-Learn and Tensorflow: Concepts, Tools and Techniques to Build Intelligent Systems”, O’Reilly, 2017
- [11] Shikha Goyal, “What are the characteristics of sound waves?”, 2018, <https://www.jagranjosh.com/general-knowledge/what-are-the-characteristics-of-sound-waves-1525678871-1>
- [12] Arsha Nagrani, Joon Son Chung, Andrew Zisserman, “VoxCeleb: a large-scale speaker identification dataset”, in arXiv, 30 May 2018
- [13] Roger Love, Donna Frazier: “Set Your Voice Free”, Little, Brown, 2016

- [14] Machine Learning Crash Course, Google
- [15] Yifan He, Zhang Zhang, “Speaker Identification with VoxCeleb Dataset”, 2017
- [16] Nguyen Nang An, Nguyen Quang Thanh, Yanbing Liu “Deep CNNs With Self-Attention for Speaker Identification”, in IEEE Access, 2019
- [17] Mark Sandler Andrew Howard Menglong Zhu Andrey Zhmoginov Liang-Chieh Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks” in arXiv, 2019