UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA

FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ ROMÂNĂ

LUCRARE DE LICENȚĂ

RECUNOAȘTEREA AUTOMATĂ A PARTITURILOR MUZICALE

Conducător științific
Conf. univ. dr. Grigoreta Sofia Cojocar

Absolvent
Răzvan Cosmin Linca

Cluj-Napoca
2020

Cuprins

1 Introducere **3**

1.1 Motivația 4

1.2 Analiza proiectului 5

2 Noțiuni introductive **6**

2.1 Pitch și clase de pitch 6

2.2 Acorduri muzicale 8

2.3 Notația prin partitură 9

2.4 Tabulatura 10

3 Metode de procesare ale semnalului sonor **11**

3.1 Semnalul sonor 11

3.2 Transformata Fourier pe termen scurt 13

3.2.1 Interpretare matematică 13

3.2.2 Aplicabilitate 15

3.3 Transformata Q constantă 16

3.4 Chromagrama 17

4 Metode de clasificare **18**

4.1 Sisteme de învățare automată 18

4.2 Modele Markov cu stări ascunse 21

4.3 Rețele neuronale artificiale 23

4.4 Rețele neuronale profunde 26

4.4.1 Rețele neuronale convoluționale 26

4.4.2 Rețele de convingeri profunde 30

5 Rezultate experimentale **31**

5.1 Descrierea soluție propuse 31

5.2 Setul de date 32

5.2.1 Augmentarea datelor 33

5.2.2 Normalizarea datelor 33

5.3 Extragerea trăsăturilor muzicale . . . 34
5.4 Detectarea tranziţiior muzicale . . . 36
5.5 Construirea, antrenarea și evaluarea modelului neuronal convoluțional . . . 38
5.6 Rezultate . . . 42
5.7 Vizualizarea graficelor . . . 43

6 Ingineria sistemului **44**
6.1 Mediu de lucru . . . 44
6.1.1 Back-end . . . 44
6.1.2 Front-end . . . 45
6.1.3 Pachete și librării externe . . . 46
6.2 Arhitectura sistemului de recunoaștere automată . . . 47

7 Abordări existente pentru recunoașterea automată a partiturilor muzicale **52**

8 Aplicație mobile **55**
8.1 Arhitectura aplicației mobile . . . 55
8.2 Funcționalități . . . 57
8.3 Proiectarea interfeței grafice . . . 60

9 Concluzii **62**
9.1 Rezumat . . . 62
9.2 Îmbunătățiri viitoare . . . 63

Capitolul 1

Introducere

Muzica acustică contemporană se află într-o perioadă de creștere din punct de vedere tehnologic, odată cu evoluția paralelă a rețelelor de socializare(platforme video și de streaming, platforme de socializare ș.a.m.d), care au conectat artiști profesioniști și amatori ai domeniului.

Această evoloție a determinat o depărtare a interpretului de partitură, acesta fiind ori un artist profesionist care învață un cântec prin simpla ascultare sau vizualizare a unei înregistrări, sau un amator pasionat care învață prin urmărirea repetată a unor tutoriale aflate pe platformele online. De altfel, este cunoscut faptul că mulți compozitori și chitariști renumiți nu puteau să citească sau să scrie partituri muzicale, bazându-se pe talentul muzical în a memora fragmente sau a improviza pe loc diverse ritmuri. Printre ei se numără Jimi Hendrix, Eric Clapton, Elvis Presley sau interpreții trupei The Beatles [39].

Chitariștii susțin că notația clasică prin partitură muzicală este foarte bine orientată către instrumente cu clape, cum ar fi pianul, fiind în general neutilizată pentru chitară. Pe de altă parte, notația prin tabulatură este optimizată pentru instrumentele cu corzi și freturi, fiind de folos mai mare pentru aceștia. Cu toate acestea, o cantitate uriașă de piese au fost scrise doar în notația clasică prin partitură, iar transcrierea unei piese de la partitură la o tabulatură corectă este un proces complex care necesită cunoștințe de specialitate. Prin urmare, materialul devine inaccesibil pentru chitariștii aspiranți care nu sunt atât de specializați.

Având în vedere existența acestor limitări și dorința chirariștilor, o soluție ar fi existența unei platforme prin intermediul căreia se pot transcrie piese acustice, în mod automat, direct într-o reprezentare prin tabulatură, specifică instrumentelor cu corzi.

Astfel, subiectul propus urmărește implementarea unei platforme pentru telefoane mobile, care să permită transpunerea automată a unor interpretări la chitară acustică, într-o reprezentare simplă și sugestivă pentru orice posibil interpret. Algorimul de automatizare va avea la bază un algoritm de învățare automată, din domeniul inteligenței artificiale. Prin urmare, implementarea acestui proces complex se bazează pe îmbinarea a două domenii aparent diferite: automatizarea unui proces ce ține de domeniul muzical și inteligența artificială, cu precădere metodele ce stau la baza învățării supervizate.

Prima parte (automatizarea recunoașterii unei partituri muzicale), definește procesul automat capabil să analize o mostră muzicală, să determine succesiunea continuă de acorduri diferite și să reprezinte automat această succesiune într-un format standard, numit tabulatură muzicală (format specific acordurilor acustice).

Analiza unei mostre muzicale și extragerea unor trăsături specifice se realizează aplicând algoritmi de procesare pentru semnalul audio, algoritmi care vor fi analizați și discutați separat și prin comparație în următoarele capitole.

Partea a doua (inteligența artificială), reprezintă unealta care stă la baza proceselor de învățare și deosebire a trăsăturilor unor acorduri din cadrul unei mostre muzicale. Se vor enunța și prezenta în detaliu câțiva algoritmi de învățare automată cu aplicabilitate pentru problema recunoașterii automate a partiturilor muzicale, cu scopul de a ajunge treptat la un algoritm complex și de actualitate pentru acest domeniu, capabil să analizeze automat semnalul audio, să clasifice cu o precizie cât mai mare fiecare secvență din cadrul unei piese acustice, fără a fi necesară intervenția umană în corectarea rezultatului

1.1 Motivația

Principala motivație a realizării proiectului are la bază dorința de a automatiza procesul de transcriere a conținutului audio direct într-o reprezentare simplă și sugestivă, folosind notația prin tabulatură. Acest proces ar trebui să fie unul simplu și rapid, sistemul fiind capabil să prezică în timp real ce acord este cântat exact în acel moment, determinând o fluiditate ridicată a sistemului, împreună cu aplicația mobile.

O motivație secundară are la bază provocarea lansată de îmbinarea dintre cele două domenii aparent diferite: muzica, prin parcurgerea unor algoritmi de procesare a semnalului sonor, și inteligența artificială, cu focus pe înțelegerea rețelelor neuronale profunde. Este o oportunitate pentru oricine abordează aceste subiecte de a acumula infomații, abilități și deprinderi noi, prin analiza și studierea inițială a domeniilor, iar apoi prin propunerea și descrierea unei soluții.

1.2 Analiza proiectului

Recunoașterea automată a partiturilor muzicale a fost un domeniu cercetat în mod activ în domeniul obținerii informațiilor muzicale, *Music Information Retrieval (MIR)*, în ultimii 20 de ani. Această categorie de algoritmi este o parte esențială a multor aplicații muzicale, cum ar fi sisteme de transcriere automată pentru diverse instrumente, aplicații de învățare în cadrul educației muzicale sau algoritmi de recomandare a muzicii sau a genurilor muzicale.

Acest domeniu urmează o paradigmă de calcul precisă care implică două faze distincte:

- Faza 1: Extragerea unor descriptori specifici din semnalul audio. Această etapă poartă numele de *extragerea trăsăturilor (feature extraction)*. Extragerea și analiza ulterioară a trasăturilor are la bază algoritmi de prelucrare și procesare a semnalului audio. Domeniul de procesare a semnalului oferă diferite reprezentări posibile ale unui semnal audio care pot fi utilizate pentru probleme de clasificare, cum ar fi problema de recunoaștere a acordurilor acustice muzicale.

  Se consideră, spre exemplu, utilizarea Transformatei Fourier în prelucrarea semnalului sonor. Procesul de extracție a caracteristicilor se încheie odată cu crearea unei reprezentări sugestive a semnalului de tip *chromagrama*. Chromagrama oferă o descriere suficient de potrivită din punct de vedere muzical a semnalului audio, fiind utilizată, în acest caz, pentru a determina cel mai probabil acord care se cântă în acel interval de timp [2].

- Faza 2: Clasificarea acordurilor pe baza reprezentării determinate în prima fază. Această etapă explorează diferite metode de clasificare cu aplicabilitate pentru problema enunțată.

  Se vor descrie și analiza în detaliu mai multe metode specifice, pe baza unor cercetări conexe ale domeniului. Astfel, se vor descrie algoritmi potriviți acestui domeniu, pornind de la modelele probabilistice, ca modelele Markov cu stări ascunse, până la construirea și optimizarea unor rețele neuronale convoluționale, utilizate recent în acest domeniu. Metoda din urmă este și cea aleasă pentru implementarea unei soluții, care va fi descrisă pas cu pas în capitolele următoare.

  Pentru a exemplifica cât mai clar și concis modul de funcționare al algoritmului, se va construi o aplicație mobile, care va fi capabilă să afișeze, în timp real, rezultatele recunoașterii automate de acorduri acustice, pentru orice partitură muzicală dorită.

Capitolul 2

Noțiuni introductive

În acest capitol sunt prezentate principalele concepte din teoria muzicală, concepte necesare în înțelegerea modalităților de prelucrare și reprezentare ale sunetelor.

2.1 Pitch și clase de pitch

Pitch-ul se definește ca atributul perceptiv care este folosit pentru a descrie percepția subiectivă a înălțimii unei note muzicale. Este atributul care permite oamenilor să organizeze sunetele pe o scară legată de frecvențe, care variază de la sunete joase la înalte [5]. Împreună cu durata, zgomotul și timbrul, este unul dintre principalele atribute auditive ale sunetelor muzicale.

Pitch-ul poate fi cuatificat ca o frecvență, dar acesta nu este o proprietate fizică pur obiectivă (precum frecvența); este un atribut psihoacustic subiectiv al sunetului. Istoric, studiul percepției pitch-ului a fost o problemă centrală în psihoacustică, fiind foarte importantă în testarea teoriilor privind reprezentarea, procesarea și percepția sunetului în sistemul auditiv [19].

Considerând astfel natura subiectivă a atributului, unii muzicieni au un simț al pitch-ului absolut sau perfect. Acest lucru înseamnă că știu întotdeauna ce notă se cântă, chiar și fără să o compare cu altă notă. Acest avantaj nu face neapărat din oricine un muzician înnăscut, dar poate fi o caracteristică foarte utilă, dacă este dezvoltată și folosită în mod corect. În decursul aniilor, mai multe teorii au încercat să explice procesul de percepție a pitch-ului de către creierul uman, dar încă nu este clar cum anume pitch-ul este codat de către creier și care sunt factorii implicați în percepție.

În domeniul muzical, fenomenul de pitch a adus în prim plan definiția conceptului de *note* care sunt asociate frecvețelor percepute. Distanța dintre două note se numește interval. Pe baza acestor noțiuni, se realizează reglarea instrumentelor muzicale (sau acordarea, dacă este vorba de instrumente cu corzi). Astfel, pentru reglare, fiecare pereche de note adiacente trebuie să aibă un raport de frecvență identic. În acest fel, distanța percepută de la fiecare notă la vecina ei este aceeași pentru toate notele posibile [5].

În acest sistem, fiecare dublare a frecvenței (octavă) se împarte în 12 părți:

$$f_p = 2^{\frac{1}{12}} f_{p-1}$$

unde f_p este frecvența unei note, iar f_{p-1} este frecvența notei precendente. Conform acestei reguli, cel mai mic interval posibil între două note corespunde raportului:

$$\frac{f_p}{f_{p-1}} = 2^{\frac{1}{12}}$$

și poartă numele de *semiton*. Relația dintre rapoartele de frecvență și intervalele muzicale este exemplificată în Tabelul 2.1.

Raport frecvențe	Interval muzical
1	Unison
2	Octavă
$2^{\frac{9}{12}} = 1.682$	Al șaselea major
$2^{\frac{8}{12}} = 1.587$	Al șaselea minor
$2^{\frac{7}{12}} = 1.498$	Al cincelea
$2^{\frac{5}{12}} = 1.335$	Al patrulea
$2^{\frac{4}{12}} = 1.259$	Al treilea major
$2^{\frac{3}{12}} = 1.189$	Al treilea minor
$2^{\frac{1}{12}} = 1.059$	Semiton

Tabelul 2.1: Legătura dintre valorile rapoartelor de frecvență și invervalele muzicale (care au o denumire specifică) [5].

Un aspect important care trebuie luat în calcul este că perpecepția umană a pitch-ului este periodică. Dacă două note sunt redate după un interval unison, care are aceeași frecvență fundamentală, pitch-ul perceput este același. De asemenea, dacă două note sunt redate după un interval la octavă, acestea determină o dublare a frecvenței, pitch-urile percepute având o calitate similară. Acest fenomenen se numește *echivalența octavei* și este adus pentru definirea conceptului de clasă pitch [5]. Oamenii sunt capabili să perceapă pitch-uri echivalente care sunt în relație de octavă.

Clasele pitch sunt clase de echivalență care includ toate notele în relație de octavă. De exemplu, clasa de pitch C (notația americană) reprezintă toate C-urile posibile, în orice poziție de octavă. Este posibilă maparea frecvenței fundamentale pentru un pitch la un număr real p folosind ecuația:

$$p = 69 + 12log\frac{f}{f_{ref}}$$

unde f_{ref} este frecvența centrală a clasei A, care are valoarea de 440Hz. Astfel, relația dintre clasele de pitch și frecvența corespunzătoare este prezentată în Tabelul 2.2.

	Octava				
Notă	2	3	4	5	6
C	66 Hz	131 Hz	262 Hz	523 Hz	1046 Hz
C#/D♭	70 Hz	139 Hz	277 Hz	554 Hz	1109 Hz
D	74 Hz	147 Hz	294 Hz	587 Hz	1175 Hz
D#/E♭	78 Hz	156 Hz	311 Hz	622 Hz	1245 Hz
E	83 Hz	165 Hz	330 Hz	659 Hz	1319 Hz
F	88 Hz	175 Hz	349 Hz	698 Hz	1397 Hz
F	88 Hz	175 Hz	349 Hz	698 Hz	1397 Hz
F#/G♭	93 Hz	185 Hz	370 Hz	740 Hz	1480 Hz
G	98 Hz	196 Hz	392 Hz	784 Hz	1568 Hz
G#/A♭	104 Hz	208 Hz	415 Hz	831 Hz	1661 Hz
A	110 Hz	220 Hz	440 Hz	880 Hz	1760 Hz
A#/B♭	117 Hz	233 Hz	466 Hz	932 Hz	1865 Hz
B	124 Hz	247 Hz	494 Hz	988 Hz	1976 Hz

Tabelul 2.2: Relațiile existente între pitch classes și frecvențe, prin aplicarea formulelor de calcul corespunzătoare [5].

2.2 Acorduri muzicale

Noțiunea de *acord* definește execuția simultană a trei sau mai multe note muzicale. Două note cântate împreună formează un interval numit interval armonic. Acordurile sunt construite combinând două sau mai multe intervale armonice în relații specifice. Cel mai obișnuit tip de acord se numește triadă, și este format din:

- nota rădăcină;
- nota caracteristică sau modală;
- nota dominantă [5].

Denumirea acordurilor se face după nota rădăcină(C – Do, D – Re, E – Mi, F – Fa, G – Sol, A – La, B – Si, fiind notația americană recunoscută internațional) și tipul lor: majore, minore, septacorduri, diminuate [30]. În repertoriul pentru chitară acustică, cele mai populare sunt acordurile majore și minore, fiind și cele recunoscute de către sistemul construit. Astfel, considerând nota de bază C, în Tabelul 2.3 se definesc cele două triade corespunzătoare acordurilor majore și minore, împreună simbolurile pentru acorduri și notele care le compun.

Triadă	Simbol acord	Note
Majoră	C, Cmaj	C E G
Minoră	Cm, Cmin	C E♭ G

Tabelul 2.3: Triada majoră și minoră pentru nota fundamentală C (Do).

După cum se observă în tabelele 2.2 și 2.3, există unele note care sunt urmate de diferite simboluri (# sau ♭). Spre exemplu, C# este echivalent cu D♭ și detemină o notă care este cu un semiton mai sus decât nota de bază C (sau cu un semiton mai jos decât D) [8].

2.3 Notația prin partitură

Notația prin partitură este cea mai frecventă formă de notare pentru instrumentele muzicale. Aceasta se preocupă cu descrierea intenției muzicale, adică o descriere a rezultatului muzical. Figura 2.1 prezintă un exemplu simplu de notație prin intermediul partiturii [28].

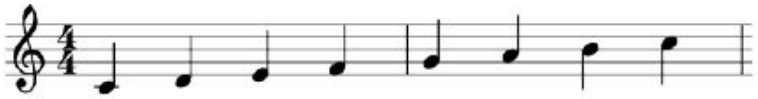

Figura 2.1: Scara ascendentă C major în notația prin partitură [28].

Acest model de prezentare a notelor muzicale este format dintr-un grilaj cu 5 linii, pe care sunt organizate notele. O notă poate să fie localizată fie pe o linie, fie între două linii. Este important de precizat faptul că această structură corespunde cu aspectul tastaturii unui pian, dar nu are o legătură directă cu modul în care sunt organizate notele în compunerea acordurilor pentru chitară [28]. Pentru a exemplica dispunerea notelor în scară, pentru notația prin partitură, Figura 2.2 exemplică toate notele muzicale într-o octavă, pornind de la nota C fundamentală. Cu acest reguli stabilite, un fragment de tabulatură este prezentat în Figura 2.2.

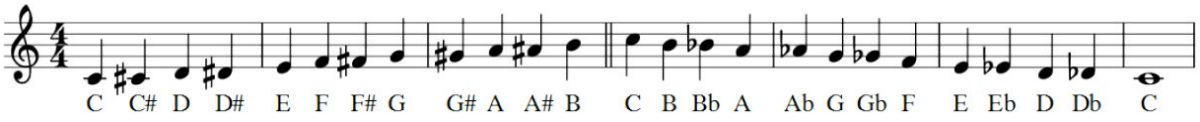

Figura 2.2: Exemplificarea notelor muzicale într-o octavă, în notația prin partitură [28].

Aceste notații stau la baza reprezentări muzicale pentru notația prin partitură. Există foarte multe alte notații, cu diferite definiții, înțelegerea și utilizarea lor fiind dependentă de studiul aprofundat al teoriei muzicale. Prin urmare, acest tip de notație nu este dedicat în mod special publicului larg, cu precădere interpreților de muzică acustică la chitară, pentru care acest tip de reprezentare a notelor muzical nu este sugestiv.

2.4 Tabulatura

Notația prin partitură descrie conținutul muzical, astfel poate să fie utilă și pentru a da diverse instrucțiuni directe despre cum trebuie abordată o piesă. Prin comparație, tabulatura surprinde la rândul ei aceste instrucțiuni și este deosebit de populară în rândul chitariștilor amatori și experimentați, prin faptul că sunt simple și eficiente pentru a fi citite sau scrise [28].

O tabulatură este formată din 6 linii orizontale care simbolizează corzile unei chitări, unde linia cel mai de sus corespunde cu cea mai de jos (cea mai subțire) coardă de pe chitară. Cifrele care apar pe tabulatură reprezintă fretul (tasta) care trebuie atinsă pentru a cânta o anumită notă. Cifra 0 sugerează faptul că acea coardă trebuie atinsă în mod liber, fără a apăsa pe niciun fret cu degetul, lăsând coarda să sune liber. Dacă cifrele stau pe aceeași poziție verticală în tabulatură, atunci acele note trebuie cântate concomitent [9]. Un fragment de tabulatură muzicală este prezentat în Figura 2.3.

```
e|--------7---------------------7-9-11-12-12-12b14---------|
B|-7-9-10---10-9-7-------7-9-10----------------------12-9---|
G|-----------------9-8-9-----------------------------------9-|
D|-------------------------------------------------------------|
A|-------------------------------------------------------------|
E|-------------------------------------------------------------|
```

Figura 2.3: Fragment de tabulatură din piesa ”And Your Bird Can Sing”, The Beatles [41].

Astfel, tabulatura este extrem de folositoare pentru a exprima relația dintre note și chitară, mai precis, dintre note și corzile unei chitări, prin specificarea poziției exacte. Acest tip se specificare este util în momentul în care sunt mai multe note care trebuie cântantate consecutiv, într-un interval scurt timp. Dacă este vorba de o interpretare acustică bazată pe acorduri muzicale, atunci se preferă reprezentarea fiecărui acord în parte, printr-o tabulatură puțin modificată. Acest tip de reprezentare este concentrată pe descrierea fiecărui acord în parte, prin specificare fret-urilor, dar și a degetelor care trebuie folosite pentru realizarea acordului muzical.

Specificarea unui acord clasic (Em) este prezentată în Figura 2.4. Aceste reprezentări se vor folosi și pentru aplicația mobile, pentru a marca fiecare acord recunoscut de către sistem.

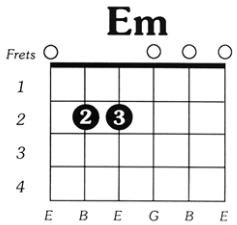

Figura 2.4: Specificarea acordului Em. Se apasă cele două corzi marcate utilizând degetul 2 (mijlociu) și degetul 3 (inelar) [12].

Capitolul 3

Metode de procesare ale semnalului sonor

În acest capitol sunt prezentate diferite metode de procesare low-level a semnalului sonor, cu scopul obținerii unor caracteristici și a unei reprezentări care va fi utilizată mai departe în definirea și antrenarea unui model neuronal, prin învățare automată.

3.1 Semnalul sonor

Sunetul, prin definiție [35], este un semnal de tip mono-dimensional, dependent de timp, reprezentând presiunea aerului asupra canalului uman auditiv. Sistemul auditiv uman este capabil să perceapă orice semnal sonor care se află în intervalul de frecvență 20Hz - 20000Hz (=20kHz).

Frecvența se definește ca numărul de repetări ale unui fenomen sau eveniment periodic într-un interval de timp. Unitatea de măsură pentru frecvență este Hertz, simbolizat ca Hz, denumită astfel în cinstea fizicianului german Heinrich Hertz. Astfel, o frecvență f = 1 Hz este corespunzătoare unei perioade de timp de o secundă.

Putem astfel afirma că dacă un anumit eveniment se repetă la un interval de timp T, putem calcula frecvența f ca:

$$f = \frac{1}{T}$$

In ceea ce privește sunetul, frecvența este legată de noțiunea de înălțime muzicală. Astfel, pentru nota LA din gama centrală se definește frecvența de 440 Hz, ceea ce înseamnă că aerul pus în mișcare de unda sonoră corespunzătoare oscilează de 440 de ori pe secundă.

Sunetul capturat de către un microfon este o undă dependentă de timp, determinând variația presiunii aerului în câmpul sonor în care se află microfonul. Astfel, un semnal audio digital este obținut prin prelevarea și cuantificarea adecvată a ieșirii microfonului, reprezentat de unde electice. Deși orice frecvență peste 40kHz ar fi suficientă pentru a capta întreaga gamă de frecvențe perceptibile, rata de preluare utilizată pe o scară largă este de 44.100 Hz, stabilită în urma nevoii de a sincroniza sunetul cu datele de tip video.

Calitatea de tip *CD* se referă la o mostră audio digitală cu frecvența de 44.1 kHz și 16-bit (Adâncimea de biți sau *Bit depth*, reprezintă numărul de biți de informație aflați în fiecare mostră audio).

Pentru a înțelege mai bine calitatea sunetului în funcție de adâncimea de biți observăm Figura 3.1, unde se prezintă prin comparație forma undei în funcție de valoarea adâncimii de biți.

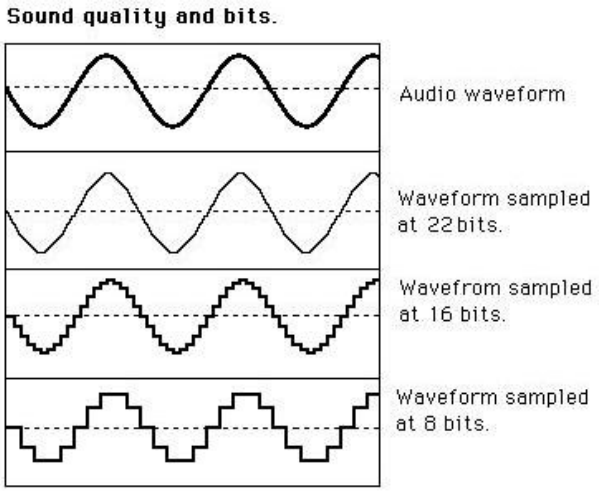

Figura 3.1: Calitatea undei sonore în funcție de adâncimea de biți [1]. Pe primul rând se observă unda sonoră originală, având 24 biți.

Prin intensitate sonoră se înțelege senzația produsă de amplitudinea unei unde sonore, cunoscută și ca volumul vibrației, asupra organului uman auditiv. Cu cât amplitudinea este mai mare, cu atât crește și intensitatea sunetului care rezultă. Intensitatea sonoră se măsoară în unități denumite în fizică decibeli(dB) sau foni (un decibel este echivalent cu un fon).

Auzul uman este limitat de un interval în ceea ce privește intensitatea sunetului, și anume:

1. Marginea inferioară:
    - prag auditiv cu valoarea de 0 dB;
    - sunete foarte slabe ca intesitate, cum ar fi: sunete din natură, nivelul de zgomot dintr-o bibliotecă, cu valorea cuprinsă între 10-20 dB.
2. Marginea superioară:
    - sunete extrem de puternice ca intensitate, cum ar fi: decolarea unui avion cu reacție, cu valorea cuprinsă între 120-130 dB;
    - prag dureros cu valorea de 140 dB [14].

3.2 Transformata Fourier pe termen scurt

Pentru a putea înțelege metoda transformatei fourier pe termen scurt, este nevoie de definirea bazelor matematice ale Transformatei Fourier (FT) și procesele care compun metoda.

3.2.1 Interpretare matematică

Se consideră, drept exemplu, un semnal pur cu frecvența f de 3Hz, pe un interval de timp vizibil. (Figura 3.2)

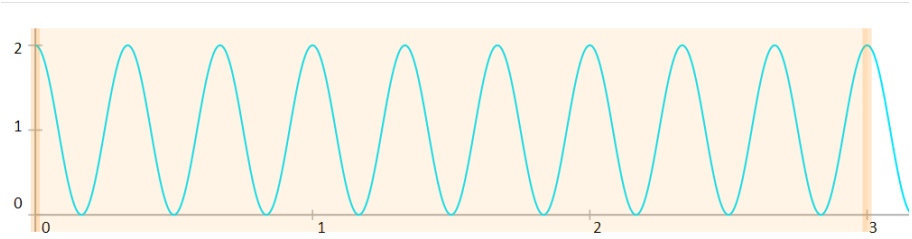

Figura 3.2: Undă sonoră cu frecvența de 3Hz [13].

Se transpune această undă dependentă de timp în jurul unui cerc prin intermediul unei corzi, în așa fel încât valorile extreme ale funcției din prima reprezentare să fie asociate cu distanțele maxime în raport cu originea din cea de-a doua reprezentare. De aici se observă că trebuie luată în considerare o nouă frecvență, și anume frecvența de realizarea a unui rotații complete de cerc parcurgând coarda, notată *fcerc*.

Se presupune că această coardă are o masă și se consideră punctul central din figură ca fiind *centrul de masă* al corzii. Se observă că odată cu modificare frecvenței cerc, se modifică și poziția centrului de masă, care se situează într-o vecinătate restrânsă a originii, cu o singură excepție: momentul în care frecvența de realizare a unei rotații în jurul cercului, *fcerc*, este egală cu frecvența undei sonore, notată *f*.

Punctul specific centrului de masă poate fi reprezentat sub forma unui număr complex de forma $xi + yj$. Se alege o reprezentare de tip complex în detrimentul unei reprezentări standard în coordonate (x,y) deoarece numerele complexe oferă o descriere potrivită pentru rotiri și ondulări raportate la un cerc de rază, notat r. De exemplu, formula lui Euler, una dintre cele mai faimoase formule trigonometrice($e^x = cos(x) + isin(x)$), enunță faptul că e^{ix} trasează cercul unitate din planul numerelor complexe, când x ia valori reale. Astfel, dacă *f=1/10*, atunci o rotație completă se va realiza odată la 10 secunde. De asemenea, convenția în cadrul transformatei Fourier este ca rotația să se realizeze în jurul acelor de ceasornic, expresia devenind $e^{-2\pi ift}$.

Fie o funcție $g(t)$ ce descrie un semnal dependent de intensitate și timp. Dacă se înmulțește $g(t)$ cu expresia construită, punctul din planul complex se va plimba de sus în jos potrivit valorilor funcției $g(t)$. Astfel, expresia $g(t)e^{-2\pi ift}$ încapsulează întreaga idee de ondulare a funcției g în jurul unui cerc cu o variabilă pentru frecvența f.

Totuși, scopul este de a urmări mișcarea centrului de masă pe care îl are coarda. Pentru a aproxima centrul de masă pot fi considerate mai multe *mostre* din semnalul original reprezentat de funcția g, apoi urmărite pozițiile lor în reprezentarea din jurul cercului, și realizată o medie. Cu cât se iau în considerare mai multe puncte, cu atât rezultatul va fi mai precis. Generalizând, se consideră o integrală a acestei funcții. De la această integrală până la transformata Fourier mai este un singur pas, și anume trebuie considerată integrala de tip continuu, pe intervalul $(-\infty, \infty)$. Astfel, formula este următoarea:

$$F(\varepsilon) = \int_{-\infty}^{\infty} f(x) \cdot e^{-2\pi i \varepsilon x} dx$$

O proprietate importantă este forma continuă a tranformatei, întrucât integrala este definită pe intervalul de timp $(-\infty, \infty)$. În practică însă, colectarea de date audio se realizează într-un interval finit de timp (de la momentul de start t_0 la momentul $t_N - 1$), ceea ce implică calculul unei transformate de tip discret, *Discret Fourier Tranform*(DFT).

Transformata de tip discret are aceeași definiție ca transformata de tip continuu, diferența fiind doar în stabilirea unui interval finit cunoscut, calcul integralei fiind înlocuit de o sumă finită care are următoarea formă:

$$X(\varepsilon_k) = \sum_{n=0}^{N-1} f(t_n) \cdot e^{-2\pi \varepsilon_k t_n}, k = 0, 1, 2..., N-1$$

Semnificația fiecărei variabile din formulă este următoarea:

- f(t_n) - semnalul de intrare, la momentul n;
- $X(\varepsilon_k)$ - valoarea complexă a spectrului corespunzător lui x, la frecvența k;
- N - totalul semnalelor de intrare (mostre);
- $t_n = nT$, unde T reprezintă intervalul de eșantionare a semnalului, fiind măsurat în secunde și n, o valoare de tip întreg, $n >= 0$;
- $f_s = \frac{1}{T}$ - reprezentând rata de eșantionare, măsurată în Hz; 1Hz = 10000 de eșantioane pe secundă, *samples per second*;
- $\varepsilon_k = k\Omega$, este al k-lea eșantion din totalul mostrelor de intrare;
- $\Omega = \frac{2\pi}{NT}$, interval de eșantionare cu frecvența radiană, rad/sec [5].

În literatură se obișnuiește ca T să fie egal 1 în ecuația precedentă si astfel se obține $t_n = n$. Atât semnalul $x(n)$ cât și transformarea $X(\varepsilon_k)$ sunt cantități discrete reprezentând N mostre. Semnalul audio este foarte non-staționar. Dezavantajul metodei DFT este că informația temporală, *temporal information*, se pierde. Din acest motiv, este necesar să se compună o analiză locală a frecvenței pe secțiuni și nu luând semnalul întreg [5]. Această metodă se numește *Transformata Fourier pe termen scurt (STFT)*. Procesul este ușor de realizat calculând DFT pe porțiuni ale semnalului, numite cadre sau frame-uri. Parametrii care trebuie specificați sunt:

- N_{FT}, dimensiunea totată a semnalului;
- $L, L <= N_{FT}$, dimensiunea unui cadru din semnal;
- saltul, *hop size*, notat Hop, reprezentând distanța în eșantioane între două cadre consecutive.

Astfel, formula de calcul pentru STFT este următoarea:

$$X_{stft}(\varepsilon_k, r) = \sum_{n=0}^{N_{FT}-1} (x(n - rHop)\varepsilon_n)e^{-2\pi\varepsilon_k n}, k = 0, 1, 2..., N_{FT}$$

3.2.2 Aplicabilitate

Transformata Fourier este o un tip de operație care se aplică unei funcții complexe și produce o altă funcție complexă care conține aceeași informație ca funcția originală, dar reorganizată dupa frecvențele componentelor.

Se dorește ca, utilizând principiile matematice ale transformatei Fourier enunțate mai sus, să se găsească o reprezentare a undei sonore, pe baza căreia se pot extrage caracteristici utile, aplicând transformata Fourier sau orice altă metodă derivată. Astfel, se consideră o funcție reprezentată de un semnal dependent de timp, care este chiar unda sonoră, dar limitată la un interval temporal. Transformata Fourier a funcției descompune semnalul după frecvență și produce un spectru al acestuia.

Proprietatea de bază a transformatei este dată de reorganizarea informației după frecvențe (temporale, spațiale sau de alt fel), fiind extrem de utilă în prelucrarea semnalelor de diverse tipuri, în înțelegerea proprietăților unui număr mare de sisteme fizice sau în rezolvarea unor ecuații și sisteme de ecuații [6]. De asemenea, transformata Fourier permite analiza semnalului și identificarea anumite frecvențe dorite, cu scopul de a le amplifica sau a le suprima, și astfel, determinându-se sintetizarea unui nou semnal.

Înzestrată cu aceste propietăți, dar și cu altele, transformata este prezentă în foarte multe tehnologii și domenii moderne(ultimul domeniu în care și-a găsit aplicabilitea fiind mecanica cuantică), deoarece este mai fiabilă și mai robustă decât alte tehnologii de analiză a semnalului de orice fel și compunere a spectrului.

Dezavantajul principal constă în dificultatea și complexitatea calculelor care determină întregul proces. În cazul integrării transformatei într-un program soft, programatorul trebuie să transpună procesul într-un algoritm eficient și să fie conștient de faptul că unele calcule utilizează procesoarele într-un mod intens.

La nivelul limbajelor de nivel înalt, cum ar fi Python, există librării care vin în ajutorul programatorilor cu metode de procesare audio implementate într-un mod eficient(de exemplu, utilizarea librăriei LibROSA).

3.3 Transformata Q constantă

Principala problemă în aplicarea Transformatei Fourier în aplicații muzicale este faptul că intervalele de eșantionare ale frecvenței pentru o mostră muzicală sunt liniar distribuite, acest aspect fiind diferit în cadrul transformatei Q constantă, unde intervalele sunt distanțate în mod geometric.

Transformarea Q constantă (CQT) a fost introdusă de Brown în [7] și este asemănătoare transformatei Fourier. Utilitatea acestei transformări se bazează pe faptul că, având un mod adecvat de alegere a parametrilor, frecvențele f_k ale transformării vor corespunde în mod direct cu cele ale notelor muzicale. Frecvențele corespunzătoare se determină astfel prin formula:

$$f_k = f_0 \cdot 2^{\frac{k}{\beta}} \ , k = 0, 1, ...$$

unde, k reprezintă a k-a frecvență, f_0 corespunde celei mai mici frecvențe iar β reprezintă numărul de filtre dintr-o octavă.

De asemenea, valoarea pentru rezoluția în domeniul timp a transformatei Q, *time resolution*, crește odată cu creșterea frecvenței. Acest aspect este similar cu comportamentul sistemului uman auditiv [5]. Concluzia acestei proprietății este că atât simțul uman auditiv, cât și computerul digital, au nevoie de mai mult timp pentru a percepe frecvența unui ton muzical scăzut.

Metoda nu este foarte eficientă în ceea ce privește timpul total de compunere a frecventelor și de determinare a reprezentări, dar reprezintă o optimizarea a tranformatei Fourier clasice, fiind utilizată pe o scară mai largă, prin aplicarea diverselor strategii de calcul paralel pentru a scădea timpul de execuție, în cazul în care tranformata este aplicată pe un set de date numeros.

Librăria LibROSA oferă o soluție eficientă și ușor de utilizat pentru calculul transformatei Q constantă a unui semnal audio, fiind nevoie doar de încărcarea semnalului și de cunoașterea parametriilor ca *hop length*, reprezentând valoare pentru salt, și opțional, *n chroma*, pentru a specifica numărul de semitonuri distincte dorite pentru octava muzicală (implicit, valoarea pentru acest parametru este 12). Astfel, o metodă simplă de calcul arată astfel:

```
def compute_chroma_cqt(audio_file_path):
    #   Se incarca fisierul audio folosind sistemul de incarcare
    # oferit de LibROSA
    audio, sr = librosa.load(audio_file_path)
    #   Determina chromagrama folosind transformarea Q constanta
    chromagram = librosa.feature.
            chroma_cqt(audio, sr=sr, hop_length=512, n_chroma=24)
```

3.4 Chromagrama

Chromagrama este reprezentarea unui grup de caracteristici pe care un program ce are la bază unul din algoritmii de procesare audio, îl extrage dintr-un semnal audio. Reprezentarea este bazată pe profilul notelor rădăcină, cunoscut ca *pitch class profile (PCP)*, acesta fiind un descriptor în contextul unui sistem de recunoaștere a acordurilor.

Chromagrama este formată astfel dintr-o secvență de vectori caracteristici, fiecare având rolul de a măsura intensitatea relativă a fiecărei note, raportată la fiecare frame, într-un interval de timp. Se obține astfel o diagramă de tipul pitch class versus time, adică o imagine a semnalului audio, ce conține informații concludente, care pot ajuta în determinarea celui mai probabil acord.

În literatură există diferite metode de a construi o chromagramă. Principalii pași enunțati de către Bello si Pickens în [2] vor fi folosiți ca bază de lucru și în procesele acestui algoritm. Pașii pentru compunerea chromagramei sunt următorii:

- Convertirea semnalului audio într-o reprezentare intermediară, numită spectogramă, prin aplicarea unei transformări Fourier derivată sau optimizată;
- Se aplică un filtru asupra frecvențelor din cadrul spectogramei, pentru a se încadra în intervalul 100Hz - 5000Hz;
- Se construiește profilul notelor rădăcină, pe baza frecvențelor estimate. Este o procedură pentru a determina nota rădăcina *(pitch class)*, din valorile frecvențelor;
- Se realizează o normalizare a caracteristicilor, cadru cu cadru, prin divizarea cu cea mai mare valoare, pentru a elimina zgomotul, rezultând imaginea chromagramei.

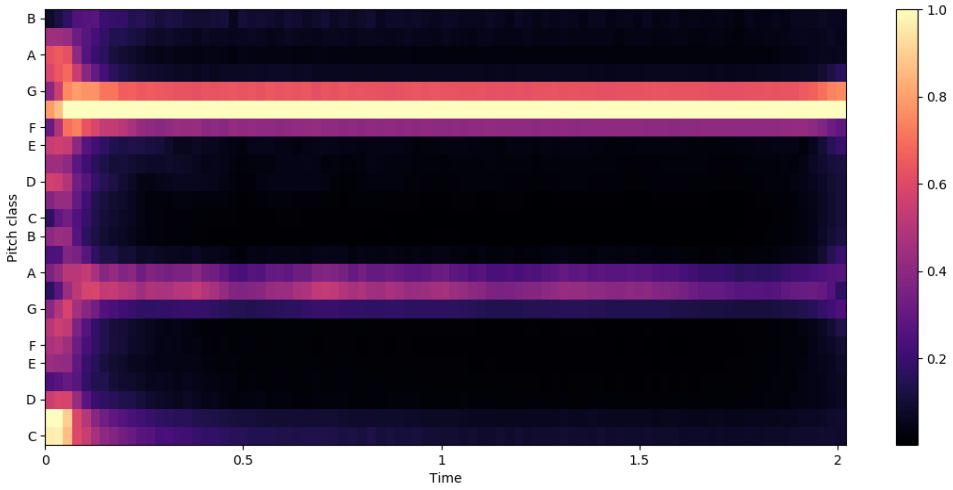

Figura 3.3: Imaginea obținută prin construirea chromagramei acordului A, reprezentare grafică utilizând LibROSA și matplot.

Capitolul 4

Metode de clasificare

4.1 Sisteme de învățare automată

Învățarea automată, *machine learning*, este o ramură a Inteligenței Artificiale concentrată pe algoritmi specializați pe invățarea din date. Procesul de învățare constă în deducerea unor tipare, pe baza unor reguli bine stabilite. Mai precis, se urmărește crearea unui program complex capabil să generalizeze un comportament indiferent de datele de intrare.

Problemele clasice de învățare automată includ clasificare de imagini sau de înregistrări audio, recunoaștere vocală, evaluarea riscurilor financiare pentru diverse investiții, dezvoltarea unor strategii în jocuri sau simulări, predicția unor diagnostice medicale etc. În mod general, abordările algorimilor sunt structurate după obiectivul lor de învățare:

1. Învățare supervizată
   În învățarea supervizată, datele de antrenament care alimentează algoritmul includ predicția corectă, numite etichete, sau *labels*. Un exemplu de metodă clasică este *metoda clasificării*.
   O problemă clasică care are la bază clasificarea este problema filter-ului de email-uri spam. Aici antrenarea se realizează cu multe exemple de email-uri, fiecare din ele având clasa corespunzătoare atașată (spam sau not-spam). Astfel, modelul va învăța cum să clasifice orice mail nou întâlnit [15].
   O altă metodă specifică este *metoda regresiei*. Problemele de regresie au ca scop prezicerea unei valori numerice target, pe baza unor trăsături numite predictori(de exemplu, prezicerea prețului pentru o mașină pe baza unor trăsături ca marca, an de fabricație și distanța parcursă). Pentru a se realiza antrenarea, este necesară utilizarea unui set de predictori împreună cu etichetele corespunzătoare.
   Cei mai importanți algoritmi de învățare supervizată sunt următorii:

   - Cei mai apropiați k vecini;
   - Modele Markov cu stări ascunse (HMMs) - *statistical learning*
   - Regresie liniară;

- Regresie logistică;
- Mașini cu suport vectorial (SVMs);
- Arbori de decizie;
- Rețele neuronale artificiale.
- Rețele neuronale profunde (CNNs, DBNs) [15].

2. Învățare nesupervizată:
   În cadrul învățării nesupervizate datele utilizate pentru antrenament nu sunt etichetate. Astfel, în funcție de date și de domeniul problemei, algoritmii sunt împărțiți în subcategorii, după cum urmează:

   (a) Clustering: se definește ca un proces de organizare a obiectelor care sunt asemănătoare dintr-un anumit punct de vedere. Astfel, noțiunea de *cluster* definește o mulțime de obiecte similare între ele și diferite de obiectele care aparțin altui cluster. Pentru exemplificare, se consideră o problemă care dorește împărțirea pe clustere a vizitatorilor unui blog. În niciun moment dezvoltatorul sistemului nu va interveni pentru a ajuta algoritmul cu informații despre vizitatori. Conexiunile între utilizatori vor fi determinate în mod independent de către algoritm [15].
   Spre exemplu, se poate observa că 30% dintre utilizatori sunt bărbați care sunt pasionați de carți polițiste și în general citesc blog-ul în jurul amiezei, în timp ce 20% sunt tineri pasionați de sci-fi si citesc blog-ul în weekend etc. Astfel, dacă se utilizează un algoritm de clustering bazat pe ierarhii, *hierarchical clustering*, se pot realiza subdiviziuni pe baza acestor informații care pot ajuta la distribuirea ulterioară în clustere.

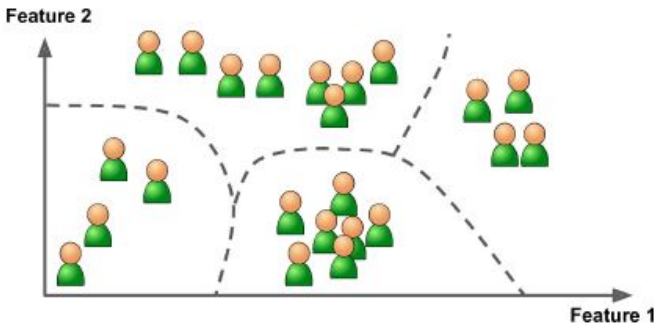

Figura 4.1: Abordarea generală a grupării în clustere [15].

   Cei mai cunoscuți algorimi de clustering sunt: *K-Means*, *Hierarchical clustering* și *Clustering bazat pe densitatea aplicațiilor cu zgomot (DBSCAN)*.

   (b) Vizualizare și reducere dimensionalității: algoritmii de vizualizare se definesc ca algoritmi care se încarcă cu numeroase date, iar pe baza lor se obține ca output un

grafic 2D sau 3D ca reprezentare a datelor, cu scopul de a înțelege cum sunt organizate acestea și posibilitatea descoperirii unor modele neașteptate. O sarcină conexă este reducerea dimensionalității, în care obiectivul este simplificarea datelor fără a pierde multe informații.
Spre exemplu, distanța parcursă de o mașină poate fi corelată cu vârsta sa, astfel algoritmul de reducere va îmbina cele două caracteristici într-o singură caracteristică ce reprezintă uzura mașinii.
Printre cele mai utilizate metode se numără: *Analiza componentelor principale(PCA)*, *Kernel PCA*, *Încorporarea linear locală(LLE)* [15].

(c) Reguli de asociere: învățarea prin intermediul regulilor de asociere are ca scop analiza cantităților mari de date și descoperirea relațiilor interesante între atribute.
De exemplu, să presupunem că deținem un supermarket. Efectuarea unei reguli de asociere pe jurnalele de vânzări dezvăluie faptul că persoanele care cumpără sos de friptură și cartofi, tind să cumpere și friptură. Astfel, este de dorit ca articolele de acest tip să fie plasate aproape unele de altele.
Printre algoritmii cei mai utilizați se numără: *Apriori* și *Eclat* [15].

3. Învățare semi-supervizată
Învățarea semi-supervizată urmărește definirea unor algoritmi de învățare care au date de antrenament doar parțial etichetate. În majoritatea cazurilor, datele neetichetate predomină în setul de date. În mare parte algoritmii sunt construiți prin combinarea algorimiilor de învățare supervizată și nesupervizată.
Spre exemplu, una din metode, rețelele *deep belief* au la bază componente nesupervizată numite mașini cu restricție Boltzmann, *restricted Boltzmann machines* (RBMs), componente așezate sub forma unei stive în definirea modelului [15]. RBM-urile sunt antrenate secvențial într-o manieră nesupravegheată, și apoi, întregul sistem este ajustat folosind tehnici de învățare supravegheată.

4. Învățare prin întărire
Învățarea prin întărire este o metodă diferită de cele enunțate anterior. Sistemul de învățare, numit agent, este capabil să observe mediul, să selecteze și să efectueze acțiuni, în urma cărora primește *recompense*. (sau penalități, sub forma de recompense negative) [15]. Acest mecanism obligă sistemul să învețe de la sine cea mai bună strategie, numită *politică*, pentru a obține cu timpul cele mai bune recompense. O politică definește ce acțiuni să aleagă agentul atunci când se află într-o situație nouă, necunoscută.

4.2 Modele Markov cu stări ascunse

Modelele Markov cu stări ascunse, *Hidden Markov Models(HMM)*, se bazează pe determinarea unui lanț Markov. Un lanț Markov este un model care ne informează despre probabilitățile secvențelor unor variabile aleatorii, numite stări, fiecare putând prelua valori din cadrul unui set. Aceste seturi pot reprezenta orice, de la cuvinte până la diverse simboluri (de exemplu, simboluri care definesc vremea).

Un lanț Markov face o presupunere foarte puternică în ceea ce privește prezicerea viitorului într-o succesiune, starea actuală fiind singura care contează. Stările anterioare stării curente nu au niciun impact asupra previziunii din viitor. Spre exemplu, s-ar putea prezice vremea de mâine, examinând vremea de astăzi, fără a fi permisă analiza vremii de ieri.

Astfel, fie o secvență de stări $q_1, q_2, ...q_i$. Un model Markov încorporează presupunerea Markov în privința probabilităților acestei secvențe. Presupunerea Markov afirmă faptul că prezicerea viitorului depinde doar de prezent, fără a conta trecutul. Matematic, se definește astfel:

$$P(q_i|q_1...q_{i-1}) = P(q_i|q_{i-1})$$

Un lanț Markov este util atunci când trebuie calculată o probabilitate pentru o secvență de evenimente observabile. În multe cazuri însă, evenimentele care sunt de interes sunt ascunse/invizibile și nu pot fi observate în mod direct. Un model Markov cu stări ascunse permite abordarea ambelor tipuri de evenimente, observabile și ascunse, considerate drept factori cauzali în modelul probabilistic.

Modelele Markov cu stări ascunse sunt caracterizate de trei probleme fundamentale:

1. Probabilitatea (*Likelihood*): Fiind dat un HMM $\lambda = (A,B)$ și un set de observații O, să se determine probabilitatea $P(O|\lambda)$;

2. Decodarea (*Decoding*): Fiind dat un set de observații O și un HMM $\lambda = (A,B)$ să se determine cea mai bună secvență de stări ascunse Q;
   Cel mai comun algoritm de decodare este algoritmul lui Viterbi, *Viterbi algorithm*. Este un algoritm de programare dinamică pentru a găsi cea mai probabilă secvență de stări ascunse, numită calea Viterbi. Rezultatul este o succesiune de evenimente observate, mai ales în contextul modelelor Markov cu stări ascunse.

3. Învățarea (*Learning*): Fiind dat un set de observații O și un set de stări într-un HMM, să se determine prin învățare parametrii A și B.
   Algoritmul standard de antrenare a unui HMM este cunoscut ca *forward-backward* sau *Baum-Welch algorithm*, fiind un caz special a algoritmului *Expectation-Maximization(EM)*. Algoritmul permite antrenarea atât a paramentrilor de tranziție (A), cât și a celor de

emisie (B). EM este un algoritm iterativ care calculează estimări inițiale pentru probabilități, apoi folosește aceste estimări pentru a calcula o estimare mai bună, procesul continuând în acest fel, îmbunătățind iterativ probabilitățile pe care le învață [21].

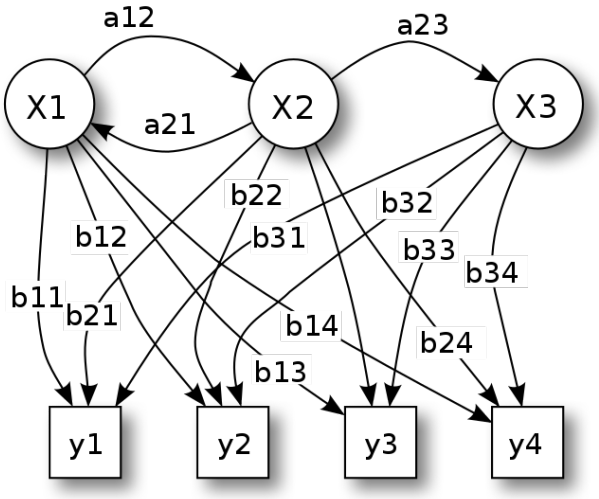

Figura 4.2: Model Markov cu stări ascunse, fiecare simbol reprezentând următoarele: X - stările, Y - posibilele observații, a - probabilitățile de tranziție între stări, b - probabilitățile de emisie [36].

Pentru recunoaștere automată a acordurilor muzicale, se aplică problema de învățare a unui model Markov cu stări ascunse, utilizând pentru antrenare vectorii care determină reprezentarea prin chromagrama. Modelul conține o singură stare pentru fiecare acord distins de către sistem.

Astfel, cunoscându-se vectorii caracteristici observați(X), etichetele pentru acorduri(Q) și parametrii modelului curent(θ), se poate calcula utilizând algoritmul EM valoarea probabilității $P(X,Q|\theta)$ pe baza unei expresii definite ulterior. Algoritmul garantează că estimările se vor îmbunătății de la o etapă la alta, ajungând într-un optim local, în ceea ce privește setul de parametrii. Astfel, soluția estimează în mod rezonabil un set de parametrii care maximizează probabilitea definită.

O astfel de abordare este descrisă în una din primele lucrări științifice ale domeniului, despre care se va discuta în Capitolul 7.

4.3 Rețele neuronale artificiale

"Birds inspired us to fly, burdock plants inspired velcro, and countless more inventions were inspired by nature. It seems only logical, then, to look at the brain's architecture for inspiration on how to build an intelligent machine [15]."

Creierul uman este cea mai complexă structură a corpului uman, fiind capabil de procesare paralelă a informației, având o capacitate de stocare a ei de 85-100 de miliarde de neuroni, fiecare având aproximativ 10000 de conexiuni.

Primul element care este astfel preluat de rețelele neuronale este neuronul. Neuronul este cea mai mică unitate fundamentală a sistemului nervos central, având rolul de a primi, conduce, procesa și transmite mai departe diferite semnale electrice primite de la organele de simț sau de la alți neuroni. Neuronul biologic este compus din [11]:

- Corp celular (Soma);
- Axon - având rolul de a transporta semnale către alți neuroni, sau celule țintă. Acesta are la capăt terminații sinaptice, prin care se poate lega cu dendritele altor neuroni sau direct cu corpul lor;
- Dendrite - prin intermediul cărora se primesc semnale de la axonii altor neuroni;
- Sinapse - reprezintă conexiuni care se realizează între axonul unui neuron și dendritele altui neuron.

Preluând aceste informații biologice, se poate realiza o mapare a fiecărui element, în vederea obținerii unei reprezentări pentru o rețea neuronală artificială.

RN Biologică	RN Artificială
Soma	Nod
Dendrite	Intrare
Axon	Ieșire
Activare	Procesare
Sinapsă	Conexiune ponderată

Tabelul 4.1: Componente: rețea neuronală biologică vs. rețea neuronală artificială

Astfel, rețelele neuronale artificiale(ANN) se definesc ca structuri artificiale care încearcă să reproducă modul de funcţionare a creierului uman. Sunt construite din mai multe unități de procesare sau *neuroni artificiali* grupaţi în straturi, fiecare strat având un număr variabil de elemente. Fiecare neuron poate primi informații de la alți neuroni, fiind acceptată chiar primirea de informații de la el însuși.

Un neuron artificial modelează comportamentul unui neuron real. Astfel conexiunile dintre neuroni, numite ponderi sinaptice, sunt folosite în stocarea informației. După o procesare locală a semnalului de intrare pe baza informaţiei stocată în ponderile sinaptice (multiplicarea acesteia cu valorile informaţionale stocate) se produce o integrare (sumare) globală a rezultatelor obţinute (proces similar cu cel ce are loc în corpul celular al unui neuron biologic real). Dacă răspunsul obţinut depăşeşte un anumit prag, informaţia este transmisă mai departe (utilizarea unei funcții de activare) [11].

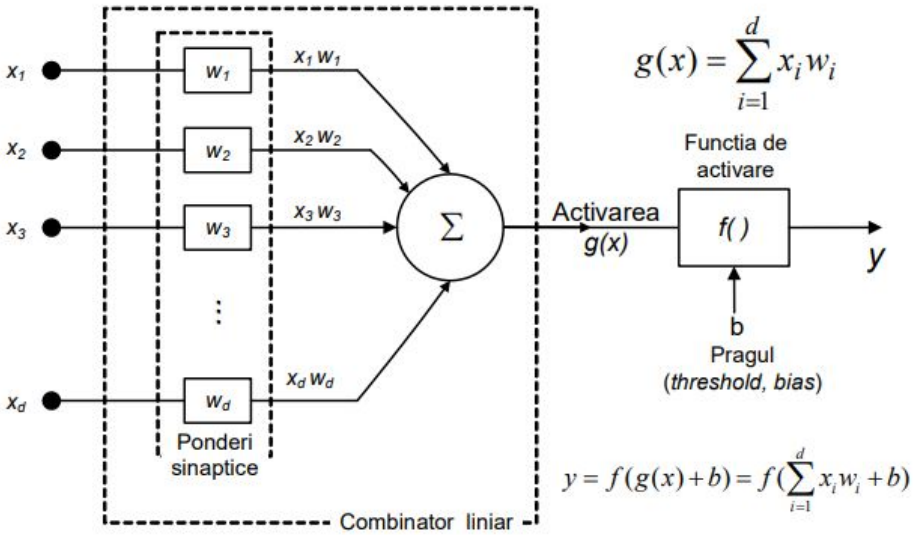

Figura 4.3: Structura generală a unui neuron artificial [11].

Valorile stocate în ponderi se schimbă prin intermediul algoritmilor de învățare. Astfel, în cadrul unui ANN, în general, cel care construiește rețeaua nu trebuie să specifice valori pentru parametrii sistemului (ponderile fiecărui neuron în parte). Valorile pentru acești parametrii vor fi extrase, în mod automat, prin intermediul algoritmilor de antrenare sau adaptare. Fiecare algoritm se clasifică în unul din obiectivele de învățare enunțate mai sus: supervizată, nesupervizată, semisupervizată sau prin întărire.

Pașii care trebuie urmați pentru construirea unei rețele neuronale artificale sunt [15]:

- Construirea stratului de intrare cu m noduri;
- Constuirea stratului de ieșire cu n noduri;
- Determinarea numărului de staturi ascunse și construirea lor, cu unul sau mai mulți neuroni pe fiecare strat;
- Inițializarea ponderilor între nodurile aflate pe straturi diferite;
- Stabilirea funcției de activare corepunzătoare fiecărui neuron (de pe straturile ascunse).

Prin specificarea tutoror elementelor enunțate mai sus, se generează *topologia* rețelei. Din punct de vedere al topologiei, ANN-urile se clasifică în două tipuri:

- Feed-Forward: Informația se propagă de la un strat la altul, de la intrare spre ieșire, iar ieșirea fiecărui nod depinde de intrările sale care sunt conectate cu ieșirile neuronilor de pe stratul precendent. Sunt folosite în special pentru învățarea supervizată.
- Recurente: Prin intermediul legăturilor existente, se creează bucle ce determină ca ieșirea diferiților neuroni să fie dependentă și de valorile anterior calculate (spre exemplu, ieșirile altor neuroni de pe același strat). Au aplicabilitate atât pentru învățarea supervizată, cât și pentru cea nesupervizată [11].

Odată ce rețeaua a fost proiectată, procesul de antrenare (învățare) poate începe. Acest proces urmărește obținerea unor valori optime pentru ponderile aflate între oricare două noduri ale rețelei, prin *minimizarea erorii*. Eroarea se calculează determinând diferența dintre rezultatul real și rezultatul calculat de rețea.

În general, rețelele neuronale artificiale sunt formate dintr-un număr mic de straturi (3-5), cele mai frecvente fiind rețelele cu 3 straturi, adică cu un singur strat ascuns. Pentru construirea unor rețele mai complexe, nu se utiliează acest tip de rețele, fiind preferate rețelele neuronale profunde (*deep neural networks*). Această categorie de rețele utilizează un număr superior de straturi, cu scopul de a dobândi capacitatea de a învăța reprezentări ale datelor de intrare cu nivele multiple de abstractizare (de exemplu, pentru clasificarea a milioane de imagini, recunoașterea audio, construirea unor sisteme de recomandare audio/video sau înțelegerea limbajului natural).

4.4 Rețele neuronale profunde

Nimic din natură sau din evoluția tehnologiei, până în ziua de azi, nu se compară cu abilitățile complexe de procesare a informației și de recunoaștere a diferitelor modele complexe pe care le are creierul uman. Încercarea tehnologiei este a avansa în această direcție, dezvoltând algoritmi care imită rețeaua creierului uman, acestea fiind numite rețele neuronale profunde (*deep neural networks*) [10].

Rețelele neuronale profunde au o structură unică, deoarece au o componentă ascunsă (formată din straturi ascunse) relativ mare și complexă între straturile de intrare și ieșire. Pentru a fi considerată o rețea neuronală profundă, această componentă ascunsă trebuie să conțină cel puțin două straturi [11]. Datorită structurii lor, rețelele neuronale profunde au o capacitate mai mare de a recunoaște tipare decât rețelele superficiale.

Există câteva arhitecturi neuronale profunde consacrate, cum ar fi:

- Rețele neuronale convoluționale (CNN)
- Rețele neuronale recurente (RNN)
- Rețele de convingeri profunde *(Deep belief networks - DBN)*

4.4.1 Rețele neuronale convoluționale

Rețelele neuronale convoluționale(CNN) sunt foarte similare cu rețelele neuronale obișnuite. Diferența constă în faptul că rețeaua face o presupunerea explicită că valorile de input sunt imagini, ceea ce îi permite să codifice anumite propietăți în cadrul arhitecturii. De asemenea, spre deosebire de o rețea obișnuită, straturile unui CNN au neuronii aranjați în 3 dimensiuni: lățime, înălțime și adâncime(de precizat, adâncimea se referă la a treia dimensiune de activare, nu la adâncimea rețelei neuronale complete, valoare care este egală cu numărul total de straturi dintr-o rețea) [10].

În general, rețelele neuronale convoluționale, cunoscute și ca ConvNets, utilizează 3 tipuri de straturi pentru a construi arhitectura, și anume: convolutional layer, pooling layer și fully-connected layer.

Un strat de convoluție *(Convolutional Layer)*, este responsabil de scanarea unei reprezentări sursă de tip imagine, aplicând un filtru de o anumită dimensiune, cu scopul de a extrage caracteristici care pot fi importante pentru clasificare. Acest filtru mai este numit nucleu de convoluție *(convolution kernel)*. Nucleul conține parametrii care pot fi ajustați pentru a atinge cele mai precise predicții.

Spre exemplu, în cadrul unui nucleu de dimensiunea 5 x 5, pentru fiecare regiune de 5 x 5 pixeli, modelul calculează o serie de produse între valorea pixelului din imagine, la o anumită poziție și parametrul corespunzător definit în filtru.

După încheierea convoluției, caracteristicile sunt comprimate *(downsampled)*, urmând ca aceeași structură convoluțională să se repete. La început, convoluția identifică trăsături din imaginea originală, apoi identifică sub-caracteristici în părți mai mici ale imaginii. În cele din urmă, acest proces este menit să identifice caracteristicile esențiale care pot ajuta la clasificarea imaginii. Straturile de convoluție produc astfel unul sau mai multe imagini numite hărți de trăsături *(feature maps)*, imagini care conțini caracteristici ce apațin imaginii originale, înainte de punerea în aplicare a nucleului de convoluție. Un exemplu de aplicare a două straturi convoluționale este prezentat în Figura 4.4.

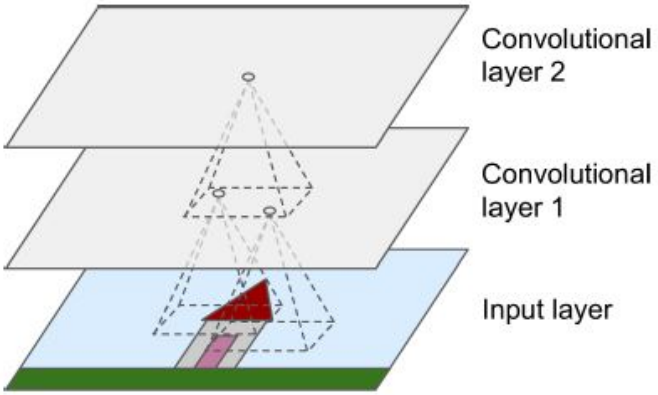

Figura 4.4: Două straturi de convoluție în cadrul unui CNN, aplicând câte un filtru pe o regiune din imaginea de input [15].

Stratul de agregare, *Pooling layer*, are rolul de a reduce/micșora dimensiunea imaginii de intrare, comprimând ieșirile unui strat convoluțional. Se consideră, ca exemplu în Figura 4.5, un filtru de dimensiunea 2 x 2 care urmează a fi aplicat folosind agregarea asupra unei imagini de dimensiunea 4 x 4. Folosind acest filtru, există două strategii care pot fi aplicate.

1. Calculul mediei valorilor din regiunea acoperită de filtru *(mean pooling)*.
2. Determinarea maximului din regiunea acoperită de filtru *(max pooling)*.

Straturile de convoluție și agregare sunt supuse unei funcții de activare cu rolul de a asigura comportamentul neliniar al rețelei. Ca funcție de activare, de cele mai multe ori se folosește ReLU *(Rectified Linear Unit)*. Mai este cunoscută ca operația de rectificare. ReLU este funcția de activare cel mai frecvent folosită în construirea modelelor de învățare profundă. Funcția returnează 0 dacă primește o intrare negativă, iar pentru orice valoare pozitivă x, se returnează aceea valoare. Pe scurt, se definește ca:

$$f_{ReLU}(x) = max(0,x)$$

O componentă complet conectată, *Fully connected*, are rolul de a realiza clasificarea propriu zisă. Această componentă este o rețea neuronală clasică, la intrarea căreia se furnizează feature map-urile, și la ieșirea căreia se aplică funcția de activare *softmax*.

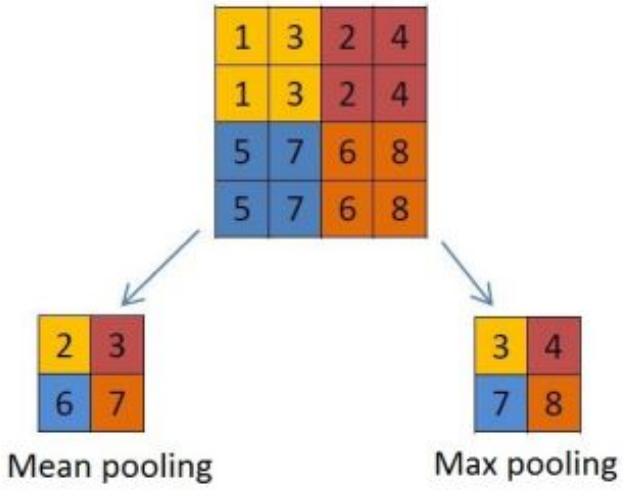

Figura 4.5: Exemplificarea operațiilor de mean pooling (stânga) și max pooling (dreapta). Fiecare element din straturile de tip pool este obținut din matricea de intrare, din zona cu aceeași culoare. În acest exemplu, din matricea de intrare de dimensiuni 4×4 se obține rezultatul cu dimensiunea 2×2 [37].

În matematică, funcția softmax, cunoscută și ca softargmax sau funcție exponențială normalizată, este o funcție care ia ca input un vector K de numere reale și îl normalizează într-o distribuție de probabilități, constând în probabilități K proporționale la valorile exponențiale ale numerelor din vectorul de intrare. Spre exemplu, înainte de aplicarea funcției softmax, unele componente vectoriale pot avea valori negative, sau mai mari decât 1, fiind posibil ca suma tuturor componentelor să nu fie egală cu 1. Dar, aplicând softmax, fiecare componentă se va afla în intervalul $(0,1)$, ceea ce înseamnă că vor putea fi interpretate ca probabilități. Funcția softmax se definește matematic astfel:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, i = 1, ..., K; z = (z_1, ..., z_k)$$

Astfel, ieșirea acestei componente este un vector de probabilități cu un număr de componente egal cu numărul de clase. Fiecare componentă a vectorului reprezintă probabilitatea ca imaginea dată ca input să se încadreze în clasa corespunzătoare.

Recapitulând întreg procesul, imaginea de la intrare conține o entitate care trebuie încadrată într-una din clasele preexistente. Input-ul este supus mai multor operații de convoluție, agregare și rectificare, de fiecare dată generându-se hărți de trăsături. Acestea conțin trăsături semnificative ale imaginii inițiale. Trăsăturile sunt apoi supuse unui proces de clasificare prin intermediul unei rețele neuronale complet conectate, rezultând o serie de probabilități ca imaginea să aparțină fiecărei categorii.

Arhitecturile tipice de CNN conțin câteva straturi convoluționale (fiecare având în continuarea lui câte un strat ReLU), apoi un strat de agregare, apoi alte câteva straturi convoluționale,

apoi un alt strat de agregare, și așa mai departe. Imaginea devine din ce în ce mai mică pe măsură ce progresează prin rețea, devenind de asemenea din ce în ce mai profundă(adică, cu mai multe hărți de trăsături), datorită straturilor convoluționale. Schița unei arhitecturi convuluționale clasice este înfățisată în Figura 4.6.

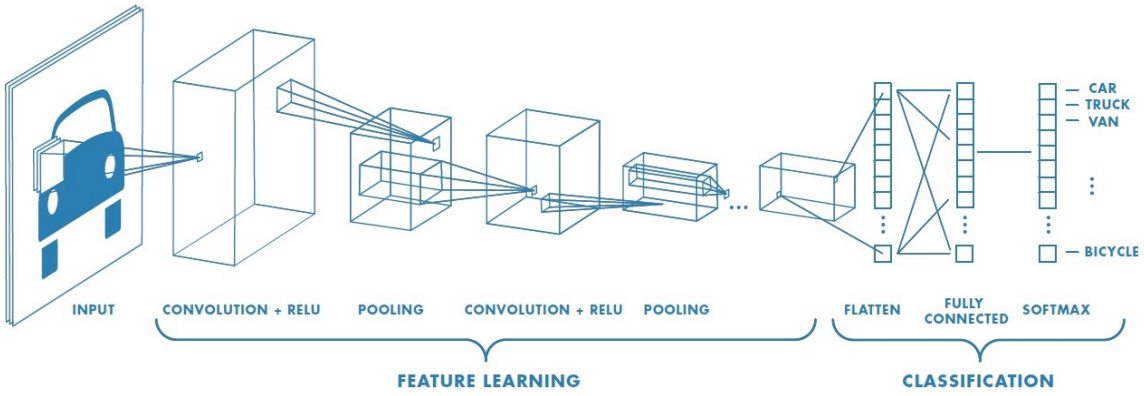

Figura 4.6: Arhitectură CNN. Se consideră o imagine ca input, care este supusă operațiilor corespunzătoare straturilor de convoluție și agregare. Rezultatul se propagă prin straturi complet conectate. Ieșirea rețelei constă într-un vector de probabilități ce încadrează imaginea într-o mulțime de clase preexistente [26].

Există o multitudine de arhitecturi actuale folosite în mod uzual, unele fiind specializate pe rezolvarea anumitor probleme. Cu toate acestea, unele sunt considerate de mare interes, fiind situate la baza acestui domeniu, printre care:

- LetNet-5 - Este cea mai veche arhitectură convoluțională, fiind descrisă pentru prima oară în 1998 cu scopul de a recunoaște cifre scrise de mână în cadrul documentelor, fiind concepută pentru imagini cu rezoluție mică;
- AlexNet - Este o arhitectură apărută după mai bine de un deceniu, în anul 2012, având 5 straturi convoluționale, acceptând imagini de intrare considerabil mai mari în rezoluție, având nuclee de convoluție mai mari în straturile inițiale (11 x 11). A fost antrenată cu ajutorul a două GPU-uri performante, fiind mult mai puternică decât arhitecturile precendete;
- VGG - Este arhitectura care a demonstrat creșterea performanței odată cu adâncimea mai mare a rețelei, având un număr semnificativ de straturi de orice tip(16, 19), fiind printre cele mai mari rețele utilizate în ceea ce privește numărul de parametri care trebuie învățați.

4.4.2 Rețele de convingeri profunde

Rețelele de convingeri profunde se prezintă ca o categorie de algoritmi care folosesc probabilitățile și învățarea nesupervizată pentru a produce rezultate. Sunt compuse din variabile latente(variabile care nu sunt observate direct, ci sunt mai degrabă deduse din alte variabile observate) binare, și conțin atât straturi nedirecționate, cât și straturi direcționate.

Spre deosebire de alte modele, fiecare strat din cadrul unei rețete de convingeri profunde învață caracteristici din întreaga entitate de input. Prin comparație, CNN-urile utilizează primele straturi pentru a filtra entitatea de intrare, obținând doar caracteristicile de bază, straturile ulterioare recombinând informația din straturile inferioare. În cadrul rețelei, nodurile din fiecare strat sunt conectate la toate nodurile din anterior si ulterior. Nodurile nu comunică lateral în cadrul aceluiași strat. De asemenea, nodurile din stratul ascuns îndeplinesc două roluri: acționează ca un strat ascuns față de nodurile care îl preced și ca straturi vizibile pentru nodurile care îl succed. Aceste noduri identifică corelații între date [44].

Pentru antrenarea rețelelor de convingeri profunde, se folosesc algoritmi de tip greedy. Această abordare este specifică algoritmilor care implică luare unei decizii la fiecare strat din secvență, găsind în cele din urmă un optim global. Învățarea se desfășoară de la un nivel la altul, ceea ce înseamnă că straturile rețelelor de convingeri profunde sunt instruite pe rând. Prin urmare, fiecare strat primește o versiune diferită a datelor și fiecare folosește ieșirea din stratul anterior ca intrare a acestora. Algoritmii de tip greedy sunt folosiți pentru a antrena rețele de convingeri profunde deoarece sunt rapizi și eficienți [44]. Mai mult, ajută la optimizarea parametriilor la fiecare strat.

În ceea ce privește ariile de aplicabilitate, acest tip de rețele pot fi utilizate cu succes în recunoașterea audio sau de imagini, dar și implementarea unor algoritmi de colectare a datelor în timp real (de exemplu, urmărirea mișcării de obiecte sau de persoane). Recunoașterea de note sau acorduri dintr-o reprezentare audio are la bază clasificarea de imagini, în urma procesării sunetului și obținerea unei reprezentări pe baza trăsăturilor extrase.

Rezultatele unei astfel de abordări vor fi prezentate în Capitolul 7, pe baza unei lucrări științifice care abordează diferite strategii, printre care și cea a rețelelor de convingeri profunde.

Capitolul 5

Rezultate experimentale

În acest capitol se va descrie soluția propusă, prin stabilirea setului de date, prezentarea arhitecturii rețelei neuronale și a etapelor intermediare construcției, și prin enunțarea rezultatelor finale obținute în urma antrenării și testării modelului.

5.1 Descrierea soluție propuse

Pentru construcția modelului de recunoaștere a acordurilor acustice, se va utiliza o rețea neuronală convoluțională. CNN-urile au crescut în popularitate în ultimii ani, mai ales în domeniul viziunii computerizate (*computer vision*).

În cadrul rețelelor neuronale convoluționale, imaginea construită sau aleasă este folosită ca parametru de intrare pentru CNN, iar imaginea este trecută prin mai multe straturi, cum ar fi straturile convoluționale, straturile de agregare și starturile de activare. Straturile convoluționale sunt construite din mai multe filtre, care pot fi interpretate individual, fiecare învățând unele caracteristici superioare ale imaginii.

În cadrul problemei de recunoaștere automată a acordurilor muzicale acustice, detectarea acordurilor poate fi tratată în mod similar cu recunoașterea imaginilor, deoarece se pot crea imagini cu reprezentarea de tip chromagramă. Identificarea notelor sau a acordurile este mai simplă decât clasificarea imaginilor, întrucât nu implică învățarea anumitor texturi, rotiri sau scalări. Cu toate acestea, această problemă vine cu alte provocări. Notele muzicale din cadrul chromagramei nu sunt localizate într-o singură regiune în același mod în care sunt cele mai multe obiecte din imagini - o notă la o anumită frecvență fundamentală va fi compusă din armonice la multiplii acelei frecvențe.

În ciuda acestei provocări, CNN-urile au proprietăți avantajoase care pot fi aplicate cu succes în această problemă. Experimentele anterioare au sugerat că agregarea informațiilor considerând mai multe cadre muzicale din același sample, determină obținerea unei predicții cu o performanță mai mare. Astfel, convoluțiile determinate asupra datelor de intrare permit modelului creat să învețe caracteristici muzicale polifonice valoroase.

5.2 Setul de date

Setul de date utilizat pentru antrenarea și evaluarea modelului convoluțional creat este format din combinarea a 3 seturi de date pentru chitară, disponibile online, asupra cărora s-au aplicat o serie de algoritmi de augmentare. Setul de date este astfel compus din:

- 7398 de acorduri individuale, grupate în 16 fișiere audio de tip wav, aparținând Institutului de semantică muzicală, Fraunhofer, din cadrul Universității tehnice Ilmenau, Germania. Setul de date se numește IDMT-SMT-CHORDS;
- 6580 de înregistrări audio, fiecare având în compoziție un singur acord, aparținând laboratorului de cercetări muzicale al Universității din New York, SUA, numit GuitarSet[34];
- 200 de fișiere audio pentru 10 tipuri de acorduri, fiind colectat de grupul de cercetare Motefiore al Universității din Liège, Belgia.

După aplicarea algoritmilor de augmentare asupra setului de date complet, s-a obținut un total de 58.577 de înregistrări audio de tip wav, grupate în cele 24+1 clase(24 de acorduri cu etichete corespunzătoare cunoscute, și o etichetă pentru orice alt acord care nu se încadrează).

Cele 24 de acorduri recunoscute de către sistem, împreună cu valorile numerice asociate sunt următoarele(perechi de forma acord muzical-valoare asociată): A-0, A#-1, A#m-2, Am-3, B-4, Bm-5, C-6, C#-7, C#m-8, Cm-9, D-10, D#-11, D#m-12, Dm-13, E-14, Em-15, F-16, F#-17, F#m-18, Fm-19, G-20, G#-21, G#m-22, Gm-23. Pentru orice alt acord care nu se încadrează în această înșiruire s-a creat perechea N-24 (acord necunoscut N, cu valoarea asociată de 24). Distribuția datelor pe clase se poate observa în Figura 5.1.

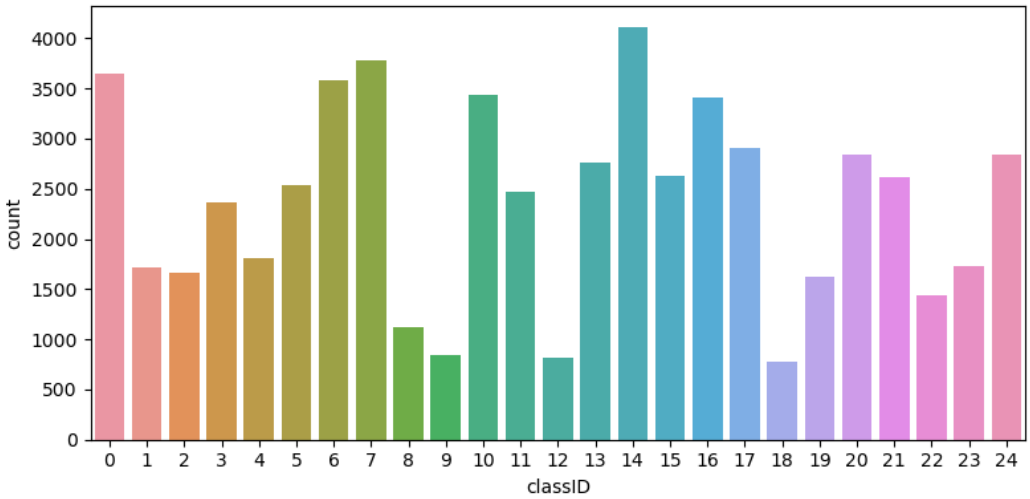

Figura 5.1: Distribuția datelor pe clase, prin specificarea numărului de fișiere audio disponibile pentru fiecare clasă asociată acordurilor muzicale.

5.2.1 Augmentarea datelor

Metodele de augmentare a datelor au rolul de a crește în mod artificial dimensiunea setului de date, generând multe variante realiste ale fiecărei instanțe set. Această procedură reduce posibilitatea de overfitting.

Instanțele generate trebuie să fie cât mai realiste, astfel încât nimeni să nu fie capabil să diferențieze între varianta originală și cea augmentată(caz ideal, în realitate vor exista aspecte care le vor diferenția) [15]. Asupra acestui set de date s-au aplicat două metode de augmentare diferite, anume:

1. Modicarea vitezei de redare, pentru fiecare instanță în parte, prin două metode:
    - Incrementarea vitezei cu 1.25% față de original;
    - Decrementarea vitezei cu 0.75% față de original;
    - Decrementarea vitezei cu 0.55% față de original (doar pentru un fragment din setul de date inițial).
2. Injectarea unui zgomot de fundal, pentru fiecare fișier audio, prin adăugarea unei valori generate random în vectorul de date specific.

5.2.2 Normalizarea datelor

Normalizarea este procesul de redimimensionare a datelor din intervalul inițial într-un alt interval prestabilit, cel mai frecvent în intervalul $[0,1]$. Algoritmul de normalizare folosit asupra setului de date este normalizarea $L-\infty$.

Normalizarea $L-\infty$ este o strategie de normalizare prin care toate valorile unui vector de caracteristici sunt aduse în intervalul $[0,1]$, prin împărțirea fiecărei valori la maximul acelui vector [5]. LibROSA oferă acest tip de normalizare ca fiind prestabilită pentru obținerea chromagramei, atât prin STFT, cât și prin tranformata Q constantă. De asemenea, librăria oferă prestabilită valoarea pentru marginea inferioară acceptată, numită *threshold*. Acest parametru are valoare 0, astfel orice valoarea sub 0 din vectorii caracteristici asociați chromagramei va fi ignorată și setată cu valoarea 0.

5.3 Extragerea trăsăturilor muzicale

Extragerea trăsăturilor muzicale are la bază unul din algoritmii de procesare a semnalului sonor, mai precis, tranformata Q constantă. Prezentarea teoretică și avantajele pe baza cărora a fost aleasă această metodă sunt prezentate în Capitolul 3.

Pentru determinarea chromagramei utilizând tranformata Q constantă s-au utilizat metodele de procesare oferite de către librăria LibROSA. Astfel, metoda aferentă transformatei primește ca parametru fișierul audio de tip wav, o valoare pentru salt sau *hop_length* egală cu 512 și valorea opțională pentru numărul de semitonuri distincte la nivelul unui octave muzicale, valoare egală cu 24.

Durata acceptată la încărcarea unui fișier audio (care reprezintă înregistrarea unui acord muzical) este de 2 secunde. Astfel, cunoscând parametrii aplicați tranformatei și durata admisă pentru un fișier audio, chromagrama construită are o dimensiune standard de 24x87, adică fiecare din cei 24 de vectori caracteristici, are 87 de valori de tip double în intervalul $[0, 1]$. În cazul în care înregistrarea are mai puțin de două secunde, se adaugă valori de 0 la vectorii caracteristici, până se ajunge la dimensiunea standard.

Pentru a eficientiza procesarea fiecărui fișier audio prin aplicarea tranformatei Q constantă, acestea vor fi procesate în paralel, prin crearea a 10 threaduri, care să execute aceeași sarcină pe intervale diferite din setul de date audio. Astfel, clasa care definește sarcinile unui thread se inițializează astfel:

```
import threading
import librosa

processed_data = []
class ChromaProcessingThread(threading.Thread):
    def __init__(self, threadID, name, start_point, stop,
                                            data_labeled):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.start_point = start_point
        self.stop = stop
        self.data_labeled = data_labeled
```

Campul *data_labeled* este un obiect construit pe baza datelor și a unui fișier Excel preprocesat, care conține informații cu privire la o înregistrare audio. Aceste informații sunt: calea absolută către acel fișier audio de tip wav, eticheta de ieșire corespunzătoare acelui fișier (numele acordului muzical) și valorea numerică asociată etichetei (cuprinsă între 0 și 24).

După inițializarea celor 10 threaduri, acestea sunt pornite, iar rularea paralelă de procesare a sunetelor și obținerea reprezentărilor prin chromagramă începe. Fiecare thread este așteptat cu o parcurgere finală, prin intermediul unor join-uri. Astfel, metoda *run*, implementată de

clasa ChromaProcessingThread, se definește în felul următor:

```
    def run(self):
        print("Starting " + self.__name + "/n")

        for i in range(self.__start_point, self.__stop):
            song, sr = librosa.load(self.__data_labeled.loc[i].
                                                    file_name, duration=2)

            # Compute chromagram
            chromagram = librosa.feature.chroma_cqt(song,
                                sr=sr,
                                hop_length=AUDIO_CONSTANTS.hop_length,
                                n_chroma=24,
                                n_octaves=7)

            # Reshape chroma matrix (if it is necessary)
            aux_chroma = []
            if chromagram.shape != (24, 87):
                for row in chromagram:
                    while len(row) < 87:
                        row = np.append(row, [0])

                    aux_chroma.append(row)
                chromagram = np.array(aux_chroma)

            # Save processed chroma matrix
            processed_data.append((chromagram, self.__data_labeled.loc[
                                                    i].classID))

        # Thread @self.__name is done
        print("Exiting /n" + self.__name + "/n")
```

După finalizarea procesării din partea celor 10 threaduri, vectorul *processed_data* ce conține toate chromagramele sub formă de matrici pentru toate fișierele audio va fi salvat într-un fișier local de tip numpy. Din acest fișier vor fi incărcate datele în etapa următoare pentru antrenarea și testarea rețelei neuronale convoluționale. Salvarea în fișierul numpy corespunzătoar se realizează prin secțiunea de cod de mai jos.

```
    import numpy as np

    np.save(AUDIO_CONSTANTS.RESOURCES_PATH +
            "//data_chroma24_hop512.npy", processed_data)
```

5.4 Detectarea tranziţiior muzicale

Detectarea tranziţiior muzicale, cunoscută ca *onset detection*, este cunoscută ca o sarcină de găsire a punctelor/vârfurilor de pornire *(peaks)*, ale tututor evenimentelor muzicale din cadrul unui semnal audio. Problema detecţie este o metodă complexă, care poate avea la rândul ei mai multe abordări, în mod simiar cu procesarea sunetului [32].

Sistemul de recunoaştere automată a acordurilor muzicale este construit şi antrenat pentru acorduri şi înregistrări individuale. Pentru a putea aplica sistemul asupra pieselor sau înregistrărilor complexe, care conţin o serie de acorduri diferite, ce se schimbă frecvent, este nevoie de a detecta cu exactitate care este momentul în care se realizează trecerea de la un acord la altul *(localization)*. Dacă se descoperă acele puncte pe axa timpului, se poate aplica algoritmul de recunoaştere pe toate intervalele determinate de punctele consecutive.

LibROSA oferă o metodă clasică de detectare a tranziţiilor, bazată pe lucrarea lui Bock, Krebs şi Schedl [4]. Metoda este compusă din 3 paşi:

- Se compune o funcţie ce denotă modificări locale ale proprietăţilor semnalului, cum ar fi energia. Este cunoscută ca funcţia de noutate spectrală sau *novelty function*;
- Se găsesc vârfurile în funcţia de noutate spectrală;
- Se aplică backtracking de la fiecare vârf la un minim local precendent, pentru a găsi punctele de segmentare, astfel încât tranziţia să apară la scurt timp după începutul segmentului [32] Figura 5.2 prezintă care sunt vârfurile evenimentelor muzicale descoperite prin aplicarea algoritmului de detecţie unei înregistări audio.

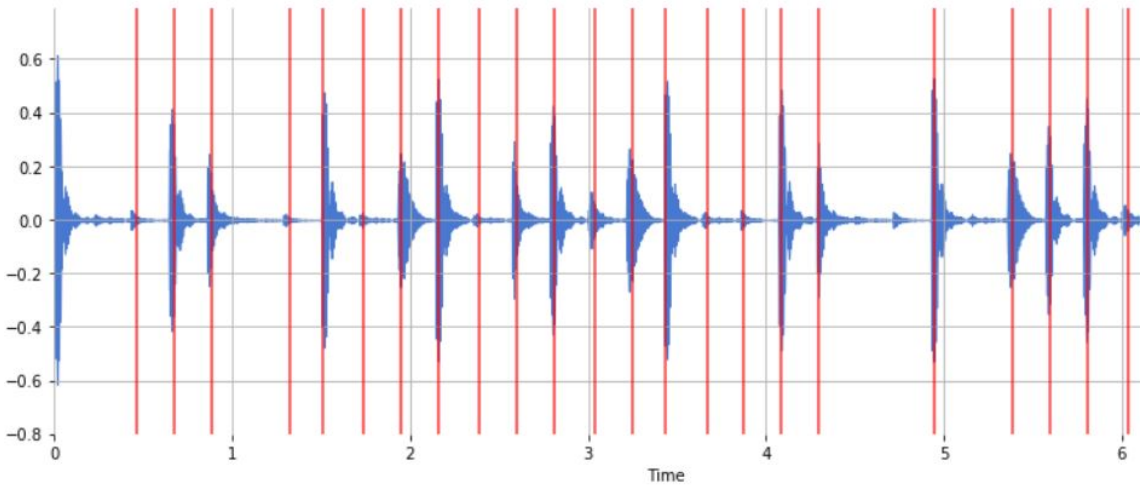

Figura 5.2: Tranziţiile muzicale detectate (reprezentate prin linii roşii) pentru o înregistrare de aproximativ 6 secunde (reprezentată sub forma unei unde sonore albastre) [32].

Întrucât segmentele determinate de tranziţiile între evenimentele muzicale descoperite în urma aplicării algoritmului sunt exacte şi marchează foarte clar care sunt intervalele de timp

cu evenimente muzicale diferite (acorduri muzicale), se aplică acest algoritm asupra pieselor sau înregistrărilor muzicale acustice, cu durata mai mare de două secunde. Astfel, aplicarea metodei de detectare a tranzițiior se realizează în felul următor:

```
    # Consider a complex audio recording
    audio_file = "//Let it Be Strum Guitar Cover.wav"
    # Load audio file
    song, sr = librosa.load(audio_file)
    # Compute the frame indices for estimated onsets in a signal
    onset_frames = librosa.onset.onset_detect(song, sr=sr,
                                              backtrack=True)
    # Convert onsets to units of seconds
    onset_times = librosa.frames_to_time(onset_frames)
```

Vectorul *onset_times* conține uniți de timp din intervalul de desfășurare a piesei audio, asupra căreia s-a aplicat metoda de detectarea a tranzițiilor. Pentru a recunoaște care sunt acordurile acestei piese, se vor evalua, prin intermediul modelului neuronal, toate intervalele de timp consecutive, pe baza unităților de timp determinate.

În urma aplicării acestei strategii, s-au observat două situații de excepție. În primul rând, dacă unul din intervalele determinate este prea scurt, modelul neuronal nu oferă predicții corecte (deoarece pe acel interval scurt, nu se extrag caracteristici suficiente), ceea ce a determinat adăugarea unei constante *beta* care să marcheze valoarea minimă pentru ca un interval să fie considerat valid pentru predicție. Astfel, dacă un interval este prea scurt, se va încerca concatenarea cu următorul interval, până când se va construi un interval valid. În al doilea rând, este posibil ca un interval să fie prea lung. Această situație este mai simplă, intervalul fiind supus divizării. Cu aceste restricții stabilite, procesarea și predicția intervalelor determinate de tranzițiile detectate se realizează astfel:

```
beta = 0.75; i = 0
while i < (len(onset_times) - 1):
    offsetValue = onset_times[i]
    diff = onset_times[i + 1] - offsetValue
    while diff < beta and i < (len(onset_times) - 1)
        and i < (len(onset_times) - 1):
            diff = onset_times[i + 1] - offsetValue
            i = i + 1
    if diff > 2:
        x = diff - 2
        diff = diff - x
    # Load the interval from original audio file
    song, sr = librosa.load(audio_file,
                            offset=offsetValue,
                            duration=diff)
    # Compute chromagram(Extragerea trasaturilor muzicale)
    prediction = model.predict_classes(np.array([chromagram]))
```

5.5 Construirea, antrenarea și evaluarea modelului neuronal convoluțional

Pentru construirea modelului convoluțional s-a folosit o secvență de straturi specializate, așezate într-o ordine bine definită pentru a spori capacitatea de învățare. Straturile utilizate sunt:

- Sequencial: Cel mai simplu model este definit folosind clasa Sequencial. Clasa definește o stivă liniară de straturi;
- ConvBlock: Definește un strat de convoluție 2D. Un strat de convoluție 2D înseamnă că input-ul operației de convoluție este tridimensioanală(3D). Cu toate acestea, acest strat este considerat a fi de tip ”2D” deoarece mișcarea filtrului peste imagine se întâmplă în două dimensiuni. Filtrul este aplicat pe imagine de trei ori, o dată pentru fiecare din cele trei straturi [44];
- BatchNormalization: Normalizarea lotului sau *batch normalization*, este o tehnică de normalizarea a activării nodurilor din straturile intermediare ale unei rețele neuronale profunde. Tendința sa de a îmbunătăți precizia și de a accelera antrenamentul, au determinat ca această tehnică să fie preferată și integrată în numeroase rețele de învățare profundă [3].
- MaxPooling: Reprezintă un strat clasic de agregare prin metoda determinării maximului din regiunea acoperită de filtru;
- Dropout: Abandonarea, cunoscută drept *dropout*, este o metodă de regularizare a rețelelor neuronale. Este o tehnică prin care anumiți neuroni selectați în mod aleator sunt ignorați în timpul antrenamentului [15]. Astfel, contribuția acestor neuroni la activarea altor neuroni este îndepărtată temporal și orice actualizarea a parametrilor în rețea nu se va aplica pentru acești neuroni.
  Efectul este că rețeaua devine mai puțin sensibilă la parametrii specifici, rezultând o rețea capabililă să generalizeze mai bine orice fel input și este mai puțin probabil ca rețeaua să se specializeze doar pentru un anumit set de date *(avoiding overfitting)*;
- Dense: Este un strat clasic de rețea neuronală complet conectată. Fiecare nod de intrare este conectat la fiecare nod de ieșire.

Astfel, cunoscând rolul fiecărui strat în parte, împreună cu funcțiile de activare ReLU și Softmax, arhitectura rețelei convoluționale construite este prezentată în Figura 5.1.

Pentru a observa starea rulării programului, dar și pentru a ține o arhivă a etapelor parcurse, în cadrul proiectul s-a integrat un sistem de logare *(logging)*, care scrie într-un fișier de logare diferite mesaje, având un format prestabilit. Instanțierea sistemului de logging s-a realizat prin metoda de mai jos.

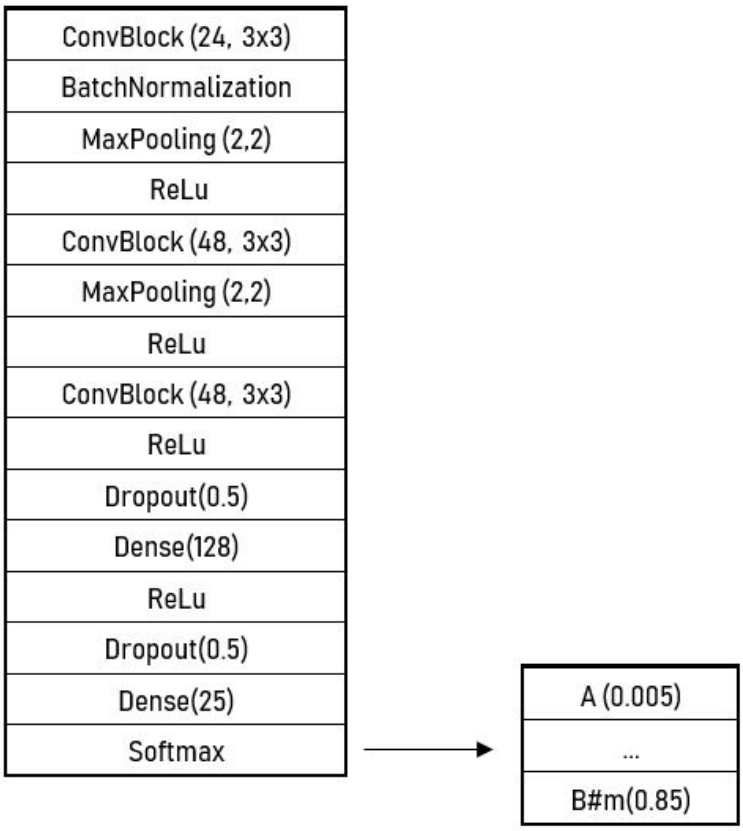

Figura 5.3: Arhitectura rețelei convoluționale construite. Numărul filtrelor și dimensiunea nucleului de convoluție sunt indicate între paranteze, pentru fiecare ConvBlock. Funcția de activare softmax este folosită în ultimul strat dens.

```
import logging
import logging.config
import yaml

def logging_setup():
    with open(LOGGING_FILE_PATH, 'r') as f:
        config = yaml.safe_load(f.read())
        # Config logging using a dictionary
        logging.config.dictConfig(config)
```

Spre exemplu, o linie din fișierul de logging arată astfel:

$2020-04-0714:52:37,717-src.modelling.cnntrain-INFO-StartTrainingProcess$

Cu setul de date procesat în urma extragerii trăsăturilor muzicale pentru fiecare fișier audio și modelul convoluțional construit, mai rămân de stabilit valorile unor parametrii înainte de startul procesului de învățare.

În primul rând, pentru antrenare, respectiv testare, setul de date va fi distribuit în mod aleator astfel: 80% din numărul total de instanțe vor fi utilizate pentru antrenare, în timp ce restul de 20% vor fi folosite pentru testarea modelului convoluțional.

Setul de date procesat a fost încărcat în memorie din fișierul numpy corespunzător, astfel:

```
import numpy as np
from science.utils import constants as AUDIO_CONSTANTS

# Load computed dataset from .npy file
dataset = np.load(AUDIO_CONSTANTS.RESOURCES_PATH
                  + "//models//data_chroma24_hop512_beta_V3.npy",
                  allow_pickle=True)
```

Pentru determinarea erorii sau a valorii pentru *loss*, s-a folosit o strategie bazată pe entropie încrucișată. Entropia încrucișată compară predicția modelului cu valoarea etichetei corespunzătoare. Valoarea ei scade, pe măsură ce predicția devine din ce în ce mai precisă. Ea devine zero dacă predicția este perfectă. Astfel, entropia încrucișată este o funcție de pierdere pentru a forma un model de clasificare.

Numărul de epoci setat este de 25. Această valoare determină de câte ori sistemul va repeta procesul de antrenare asupra datelor corespunzătoare, acesta învățând de la o etapă la alta, devenind tot mai precis, cu o eroare din ce în ce mai mică.

În ceea ce privește algoritmul de optimizare, s-a ales algoritmul de optimizare Adam, cu dimensiunea unui bloc de 64 *(batch_size)*. Adam este o metodă de adaptare a ratei de învățare, ceea ce înseamnă că rata de învățare este calculată individual pentru diferiți parametri [15].

Întrucât librăria folosită pentru construirea modelului este Keras, se acceptă setarea ca parametru a unei instanțe pentru vizualizarea unor rezutate intermediare în timpul antrenării. Instanța este un obiect de tip TensorBoard, și se aplică modelului prin parametrul *callbacks*. De asemenea, tot ca parametru se poate seta o instanță pentru monitorizarea rezultatelor. Astfel, asupra modelului s-a aplicat o instanță de tip *EarlyStopping* care are rolul de a opri antrenamentul în momentul în care una din valorile monitorizate a încetat să se îmbunătățească.

Cu toate aceste valori cunoscute, înțelese și setate corespunzător pentru rețeaua neuronală convoluțională, antrenarea poate începe cu succes. Metoda de antrenare utilizată în proiect este afișată în secțiune de cod de mai jos.

```
import keras
import logging
import os

def train(self, X_train, Y_train, X_test, Y_test):
    logger.info("Start training cnn model")

    # Compile model
    self.model.compile(
        optimizer="Adam",
        loss="categorical_crossentropy",
        metrics=['accuracy', precision, recall, f1measure])
```

```
        # TensorBoard Logging
        log_dir = os.path.join(AUDIO_CONSTANTS.LOGGING_PATH, datetime.
                                            datetime.now().strftime("%Y
                                            %m%d-%H%M%S"))
        tensorboard_callback = keras.callbacks.TensorBoard(log_dir=
                                            log_dir, histogram_freq=1)
        logger.info("Tensorboard Logging Started")

        # Start training
        self.__history = self.__model.fit(
            x=X_train,
            y=Y_train,
            epochs=25,
            batch_size=64,
            validation_data=(X_test, Y_test),
            callbacks=[tensorboard_callback,
                       EarlyStopping(monitor='val_loss',
                       verbose=1, patience=2)])
        logger.info("Training completed")
```

După finalizarea antrenării, s-a realizat evaluarea modelului neuronal convoluțional pentru a calcula valorile metricilor la antrenare, dar cel mai important la testare, asupra unor date noi, care nu au mai fost ”văzute” de către sistem. Valorile lor au fost salvate prin sistemul de logging al proiectului, urmând a fi preluate din fișierul de logări pentru concluzii. Secțiunea de cod care realizează evalurea este afișată mai jos.

```
    def evaluate(self, X_train, Y_train, X_test, Y_test):
        self.score_train = self.model.evaluate(
            x=X_train,
            y=Y_train)

        self.score_test = self.model.evaluate(
            x=X_test,
            y=Y_test)

        logger.info(f'Train loss: {self.score_train[0]}')
        logger.info(f'Train accuracy: {self.score_train[1]}')
        logger.info(f'Train precision: {self.score_train[2]}')
        logger.info(f'Train recall: {self.score_train[3]}')
        logger.info(f'Train f1-score: {self.score_train[4]}')

        logger.info(f'Test loss: {self.score_test[0]}')
        logger.info(f'Test accuracy: {self.score_test[1]}')
        logger.info(f'Test precision: {self.score_test[2]}')
        logger.info(f'Test recall: {self.score_test[3]}')
        logger.info(f'Test f1-score: {self.score_test[4]}')
```

5.6 Rezultate

Metricile utilizate au fost acurateţe, precizie, recall şi scorul F. Pentru a defini mai clar modul de calcul al acestor măsurători, se vor introduce următoarele variabile:

- True Positives (TP): Numărul de cazuri cu predicţia DA, şi valoarea reală DA;
- True Negatives (TN): Numărul de cazuri cu predicţia NU, şi valoarea reală NU;
- False Positives (FP): Numărul de cazuri cu predicţia DA, şi valoarea reală NU;
- False Negatives (FN): Numărul de cazuri cu predicţia NU, şi valoarea reală DA.

Metricile alese au valori relevate doar în cazul în care există un număr echilibrat de probe pentru fiecare clasă în parte. De exemplu, se consideră existenţa a 98% probe din clasa A şi 2% eşantioane din clasa B în setul de antrenament. Modelul obţinut poate obţine cu uşurinţă acurateţea de 98% prezicând corect fiecare probă aparţinând clasei A. Valoarea nu este însă una reală, modelul nefiind pregătit să recunoască instanţe ale clasei B.

În cazul setului de date utilizat această problemă nu există, fişierele audio fiind distribuite în mod echilibrat în cele 25 de clase. Cunoscând măsurătorile auxiliare şi proprietatea de echilibru în distribuţia datelor, metricile utilizate pot fi folosite cu succes. Acestea se definesc prin formulele de mai jos:

$$Acuratete(A) = \frac{TP+TN}{TP+TN+FP+FN}$$

$$Recall(R) = \frac{TP}{TP+FN}$$

$$Precizie(P) = \frac{TP}{TP+FP}$$

$$Scor-F = \frac{2 \cdot R \cdot P}{R+P}$$

Aceste metrici au fost aplicate la compilarea modelului neuronal convoluţional. Rezultatele obţinute sunt prezentate în Tabelul 5.1.

Etapă	Acurateţe	Precizie	Recall	Scor-F
Antrenare	97.23%	98.38%	95.85%	97.05%
Testare	89.75%	93.66%	86.67%	89.98%

Tabelul 5.1: Rezultate obţinute, în urma celor două etape: antrenare şi testare.

5.7 Vizualizarea graficelor

Pentru a observa evoluția proceselor de antrenare și testare, și impactul întregului proces asupra metricilor de calcul, s-au realizat două grafice pentru a evidenția scăderea loss-ului și creșterea acurateții. Astfel, Figura 5.3 prezintă valorile pe care le-a avut eroarea pe parcursul celor 25 de epoci. Se observă o scădere relativ exponențială, valorile loss-ului la antrenare, respectiv la testare fiind apropiate. Figura 5.4 prezintă valorile pentru acuratețe, fiind evidențiat un trend ascendent, valorile fiind apropiate spre ultimele epoci.

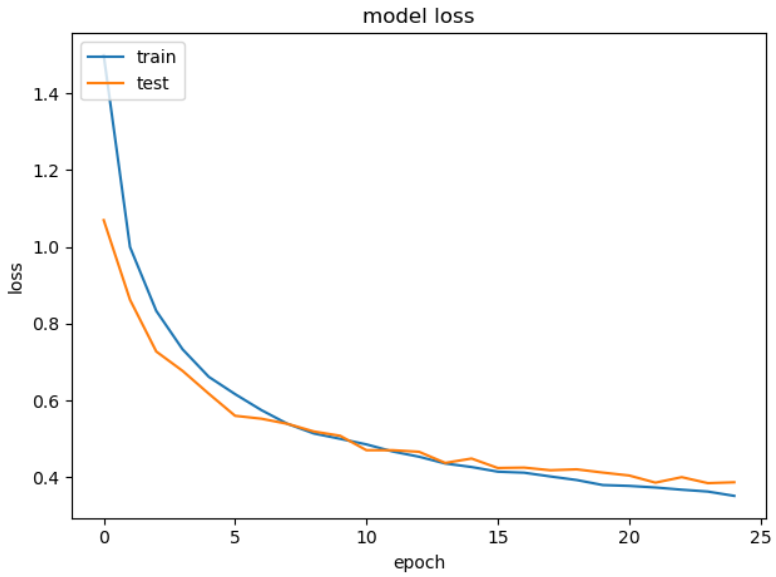

Figura 5.4: Evoluția valorii pentru loss, la antrenare și testare.

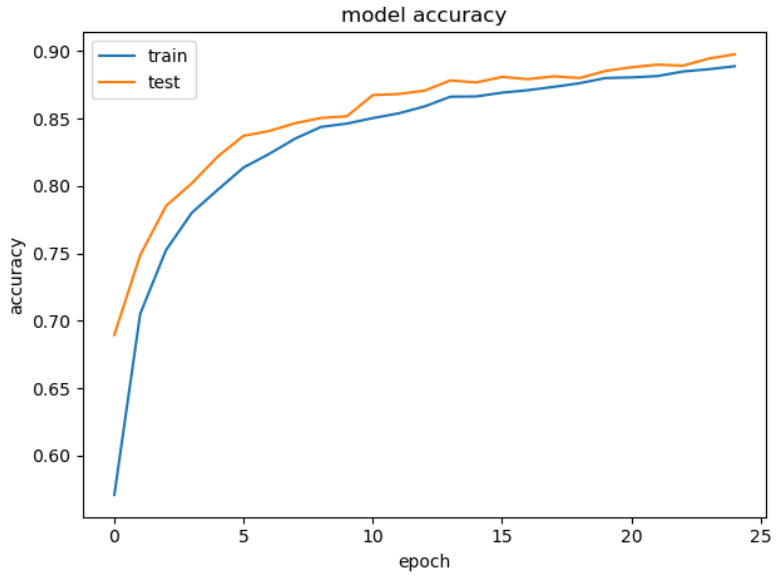

Figura 5.5: Evoluția valorii pentru acuratețe, la antrenare și testare.

Capitolul 6

Ingineria sistemului

Acest capitol urmărește prezentarea proiectului din perspectiva ingineriei de sistem, analizând limbajele de utilizare și mediile de dezvoltare folosite, dar cel mai important, prezentând arhitectura sistemului, prin evidențierea modelelor software, atât la nivel teoretic, cât și practic, prin construirea diagramelor specifice.

6.1 Mediu de lucru

6.1.1 Back-end

Limbajul de programare folosit pentru realizarea proiectului de back-end este Python. Este un limbaj de de nivel înalt, orientat obiect, ce pune accent pe expresivitatea și înțelegerea ușoară a codului. Constucția sa la nivel înalt, în combinație cu tastarea dinamică și legarea dinamică, îl fac foarte atractiv pentru dezvoltarea rapidă a aplicațiilor, precum și pentru utilizarea în scripting sau ca limbaj de legătură pentru a conecta componentele existente împreună [43].

Python se caracterizează prin lizibilitate și, prin urmare, reduce costurile de întreținere a unui program. Este construit să suporte module și pachete, care încurajează modularitatea programului și reutilizarea codului. Interpretorul Python și biblioteca extinsă standard sunt disponibile în formă sursă sau în formă binară, fără taxe, pentru toate platformele majore, fiind posibilă distribuirea în mod liber [43].

Python este unul din limbajele favorite în rândul dezvoltatorilor, fiind potrivit pentru multe tipuri de aplicații. În mod particular însă, este considerat ca fiind cea mai bună alegere pentru proiectele care implică inteligența artificială. Acest lucru este datorat faptului că limbajul conține foarte multe biblioteci și framework-uri externe, care facilitează procesul de codare și economisește timpul de dezvoltare. Învățarea automată și învățarea profundă sunt foarte bine tratate, prin numeroase librării printre acestea se numără NumPy, folosit pentru calcul științific, SciPy pentru calcule avansate și scikit-learning pentru minerirea datelor și analiza datelor, fiind printre cele mai populare biblioteci, lucrând alături de framework-uri externe puternice precum

TensorFlow, Keras, CNTK sau Apache Spark.

Atât învățarea automată cat și cea profundă se bazează pe algoritmi extrem de complecși și pe fluxuri de lucru în mai multe etape. Prin urmare, cu cât un dezvoltator trebuie să se îngrijoreze mai puțin de complexitatea codificării, cu atât mai mult se poate concentra pe găsirea unei soluții optime și atingerea obiectivelor proiectului. Sintaxa simplă în Python determină o dezvoltare mai rapidă decât alte limbaje de programare și permite dezvoltatorului să testeze rapid algoritmii fără a fi nevoie să îi implementeze.

În ceea ce privește mediul de dezvoltare (IDE), au fost utilizate două medii diferite, în funcție de complexitatea proiectului. În primă fază, pentru realizarea unui prototip de model neuronal și acumularea de deprinderi utilizând librării ca TensorFlow, Keras, NumPy sau LibROSA, s-a utilizat Google Colab. Acesta este un mediu de tip *notebook*, care rulează în întregime în cloud. Cel mai important aspect în ceea ce îl privește este că nu necesită configurare, iar fiecare filă de lucru care se poate crea poată fi rulată și editată în mod simultan, similar cu documentele din suita Google Docs [16]. Colab este configurat să accepte biblioteci populare de învățare automată sau profundă, care pot fi încărcate cu ușurință în notebook.

Odată cu creșterea proiectului în complexitate, utilizarea foilor de lucru oferite de Google Colab nu a mai fost suficientă, iar modularizarea sistemului a devenit o necesitate. Astfel, mediul de dezvoltare a fost schimbat, fiind ales editorul PyCharm. PyCharm este un IDE cu caracteristici complete pentru limbajul Python, bazat pe puternica platformă Java IntelliJ Idea de la JetBrans. Nu numai că PyCharm acceptă o serie de funcții la nivel enterprise, dar este, de asemenea, o plăcere de a lucra cu el și în modul community [18]. Pentru mulți dezvoltatori în domeniul inteligenței artificiale, PyCharm este următorul pas după mediile de învățare precum notebook-urile iPython sau Google Cobab și este o alegere populară pentru dezvoltarea aplicațiilor full-stack.

6.1.2 Front-end

Pentru dezvoltarea proiectului de front-ent, limbajul de programare utilizat este Kotlin. Este un limbaj de programare de tip open-source, care suportă atât programarea orientată obiect, cât și programarea funcțională. Kotlin oferă sintaxă și concepte similare altor limbaje, precum C#, Java sau Scala [22]. Kotlin nu își propune să fie unic, ci se inspiră din dezvoltarea limbajelor similare de-a lungul deceniilor. Există în variante care vizează JVM (Kotlin/JVM), JavaScript (Kotlin/JS) și cod nativ (Kotlin/Native).

Anumite API-uri pentru Android, cum ar fi Android KTX, sunt specifice Kotlin, dar majoritatea sunt scrise în Java, și pot fi apelate fie din Java, fie din Kotlin. Interoperabilitatea dintre Kotlin și Java este esențială pentru creșterea limbajului. Această proprietate îi permite programatorului să apeleze la cod Java într-un sistem dezvoltat în Kotlin, și viceversa. Popularitatea lui Kotlin are ca rezultat o experiență de dezvoltare mai bună pe Android, dar dezvoltarea mediului în Android continuă atât cu Kotlin, cât și cu Java [22].

În ceea ce privește mediul de dezvoltare ales pentru construirea aplicației mobile, s-a utilizat Android Studio. Acest IDE este mediul oficial a sistemului de operare Android, bazat pe software-ul IntelliJ IDEA de la JetBrains. Android Studio este disponibil pentru Windows, OS X și Linux. Pe data de 17 mai 2017, la conferința anuală Google I/O, s-a anunțat asistență pentru limbajul Kotlin, ca limbaj de programare oficial pentru platforma Android [27].

6.1.3 Pachete și librării externe

Construcția întregului sistem a urmat o serie de etape clare, care au condus în final la furnizarea unei aplicații complet funcționale, pornind de la funcționalitățile oferite utilizatorului prin aplicația mobile, până la construirea unui model neuronal convoluțional și punerea acestuia în funcțiune. Toate acestea ar fi fost realizate mult mai greu, dacă nu ar fi existat librăriile externe utilizate. Printre ele se numără:

- LibROSA: librărie Python pentru analiză audio. Furnizează pachetele necesare construirii unui sistem de obținere a informațiilor muzicale [24];
- TensorFlow: platformă open source de tip ”end-to-end” destinată învățării automate. TensorFlow este un sistem bogat pentru gestionarea tuturor aspectelor unui sistem de învățare automată. API-urile TensorFlow sunt aranjate ierarhic, cu API-uri de nivel înalt construite pe API-uri de nivel scăzut [25]. Cercetătorii în domeniu folosesc API-uri de nivel scăzut pentru a crea și a explora noi algoritmi de învățare;
- Keras: este un API de nivel înalt, orientat pe rețele neuronale, scris în Python, fiind capabil să ruleze pe TensorFlow, CNTK sau Theano [25]. Acesta a fost dezvoltat cu focus pe realizarea experimentelor într-un mod cât mai simplu și rapid. Se recomandă Keras pentru următoarele:
  - crearea unor prototipuri simple, extrem de rapid;
  - suport pentru rețele neuronale convoluționale și rețele recurente, cât și combinații între cele două;
  - rulare pe CPU sau GPU, întrucât eficiența este aceeași.
- NumPy: este biblioteca principală pentru calcul științific în Python. Oferă un obiect de tip matriceal multidimensional de înaltă performanță și instrumente de lucru cu acest tip de tablouri, printe care o colecție mare de funcții matematice [31];
- Pandas: este un pachet Python care oferă structuri de date rapide și flexibile, concepute pentru a face lucrul cu datele structurate (tabulare, multidimensionale, potențial eterogene) cât mai ușor și intuitiv [33].

6.2 Arhitectura sistemului de recunoaștere automată

Arhitectura unui sistem surprinde structura sistemului în ceea ce privește componentele acestuia precum și modul în care aceste componente interacționează între ele. Arhitectura trebuie să evidențieze și modalitățile de interacțiune ale utilizatorilor cu sistemul, prin specificarea modulelor responsabile de interacțiune [40].

Pentru o descriere cât mai specifică a acestor aspecte, se vor evidenția prin realizarea unor diagrame, mai multe tipuri de modele software, cu aplicabilitate pentru sistemul în cauză. Aceste modele se împart în 3 categorii:

1. Modelul funcțional;
2. Modelul dinamic;
3. Modelul obiectual.

Limbajul care stă la baza diagramelor unui sistem este limbajul UML *Unified Modeling Language)*, fiind limbajul standard adaptat și recunoscut în industrie pentru reprezentarea modelelor orientate obiect. Este considerat un limbaj de modelare de tip vizual, cu accent pe modelare, fără a avea conexiuni cu proprietățile unui limbaj de programare vizual, fiind folosit ca un limbaj universal pentru descrierea sistemelor software [40].

Modelul funcțional are rolul de a descrie funcționalitatea sistemului din punct de vedere a interacțiunii cu utilizatorul. Prezentarea modelului se face prin intermediul unui diagrame a cazurilor de utilizare. Cazurile de utilizare sunt folosite în cadrul activităților de colectare și analiză a cerințelor problemei, cu scopul de a reprezenta funcționalitățile sistemului. Elementele UML conţinute într-o astfel de diagramă sunt: cazuri de utilizare, actori, relaţii de includere, relaţii de extindere, relaţii de generalizare şi de asociere.

Elementele care compun o diagramă a cazurilor de utilizare au rolul de a defini comportamentul sistemului, a subsistemelor sau a unei entităţi, fără a specifica structura internă [40]. Este o generalizare, cu scopul de a vizualiza toate funcționalitățile, din perspectiva utilizatorului, înainte de a intra în analiza tehnică a acestora. Diagrama cazurilor de utilizare prezintă relația dintre actori, cazuri de utilizare și sistem. Figura 6.1 prezintă diagrama cazurilor de utilizare aferente sistemului de recunoaștere automată, împreună cu funcționalitățile conexe.

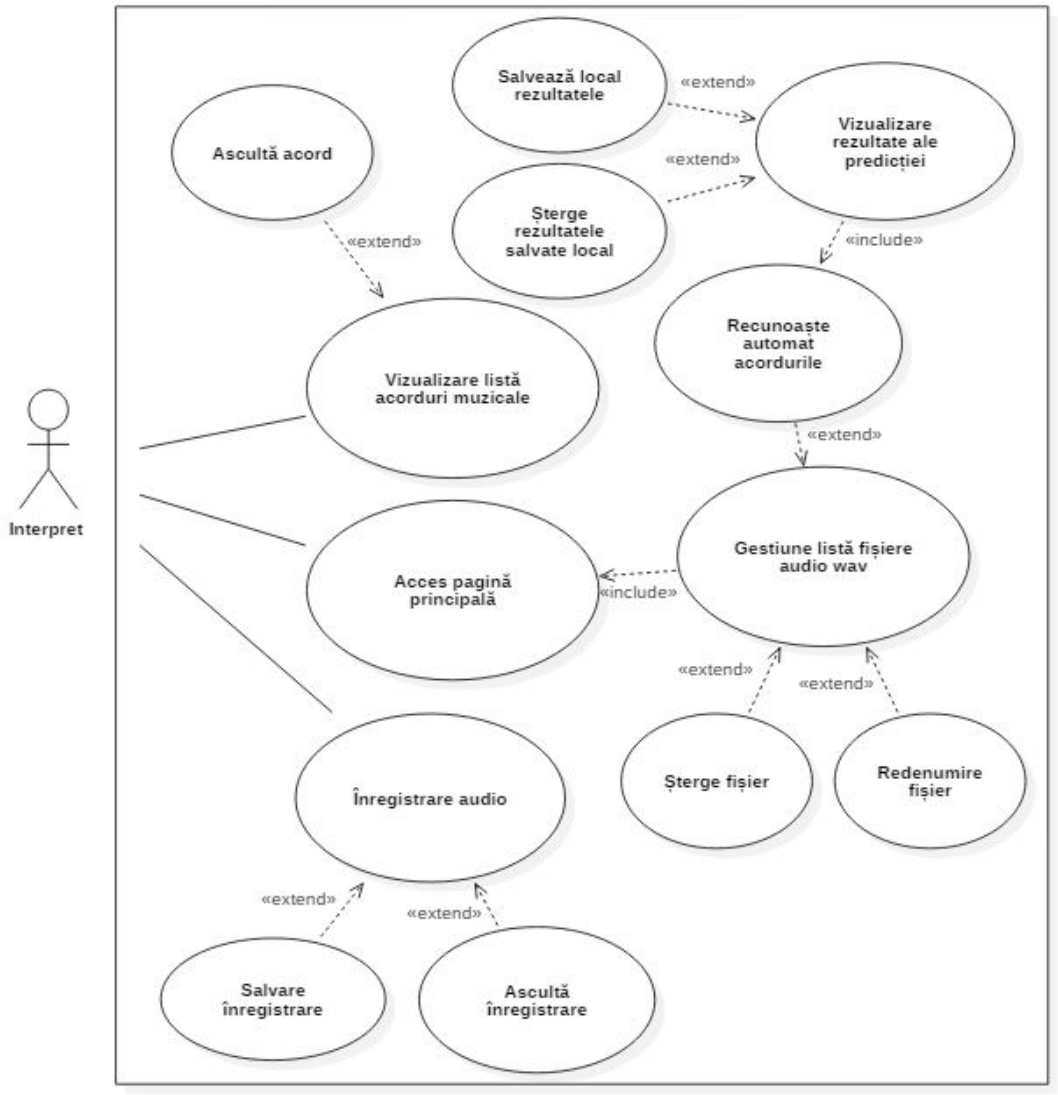

Figura 6.1: Diagrama cazurilor de utilizare, ilustrând actorul ca ”Interpret” și fluxul de evenimente între actor și sistem.

Un alt model important este cel dinamic, care descrie modul în care grupuri de obiecte colaborează în cadrul aceluiași eveniment. Specificarea modelului se face prin intermediul unui diagrame de interacțiune. Există două tipuri de diagrame de interacțiune: diagramele de secvență și diagramele de comunicare. Cele două tipuri de diagrame sunt echivalente, astfel cunoscându-se o diagramă de secvență, se poate construi diagrama de comunicare, și reciproc [40]. Pentru exemplificare, se va utiliza diagrama de secvență.

Diagramele de secvență au rolul de a reprezenta interacțiunile între obiecte. Obiectele participante sunt desenate sub formă de dreptunghiuri, fiind reprezentate pe orizontală, axa timpului fiind pe verticală. Coloana cea mai din stânga corespunde actorului care declanșează acțiunea. Acțiunea care este declanșată trebuie să corespundă cu un caz de utilizare definit în prima etapă [40]. Figura 6.2 prezintă diagrama de secvență aferentă cazului de utilizare corepunzătoar procesului de recunoaștere automată a unei partituri muzicale dintr-o piesă acustică.

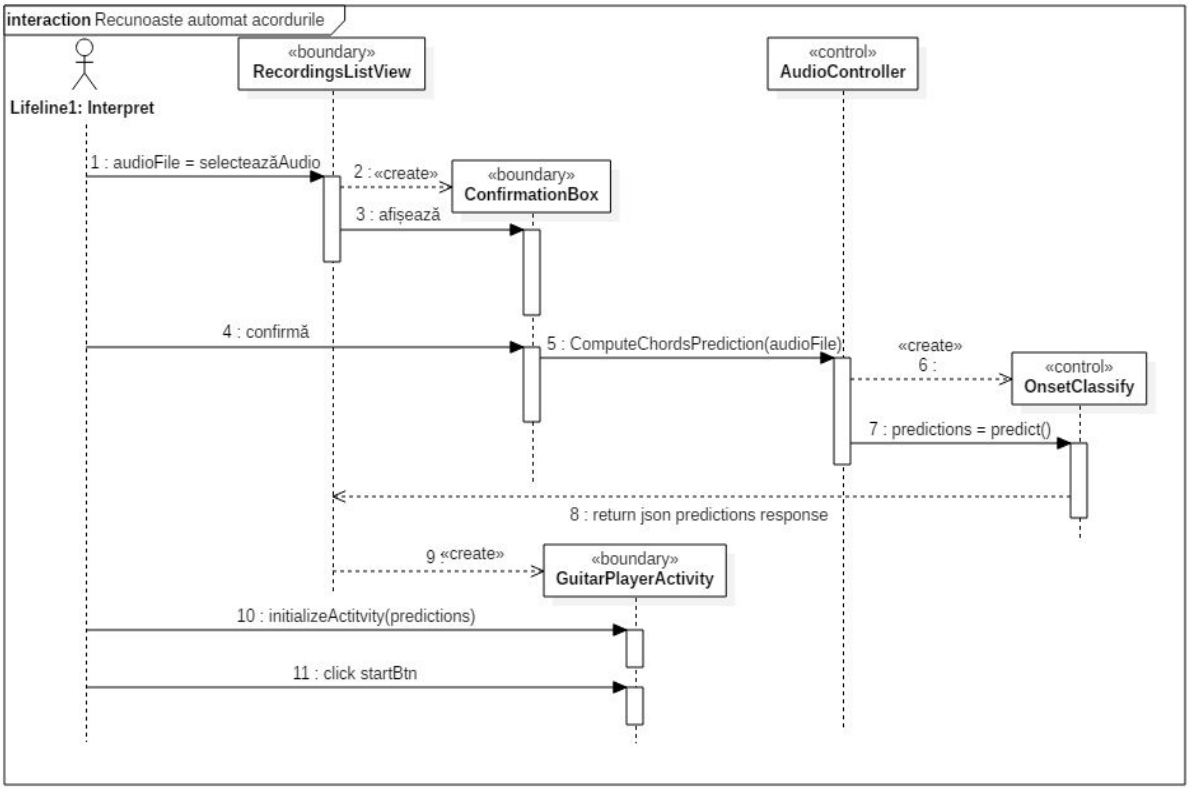

Figura 6.2: Diagrama de secvență a cazului de utilizare ”Recunoaște automat acordurile”.

Ultimul model, modelul obiectual, descrie structura sistemului apelând la obiecte, atribute, asocieri și operații [40]. Diagrama de clase este implicată în descrirea modelului, fiind compusă dintr-un graf, ale cărui noduri reprezintă clasele sistemului construit, și ale cărui arce determină relațiile între clase. Clasele sunt reprezentate în UML folosind dreptunghiuri cu 3 compartimente (numele clasei, atributele și operațiile), în timp ce arcele pot fi de mai multe tipuri, în funcție de specificul legăturii (asociere, agregare, generalizare, compoziție).

Diagrama de clase aferentă sistemului construit cu scopul automatizării procesului de detectare și recunoaștere a acordurilor muzicale este prezentată în Figura 6.3.

Se observă că structura proiectului, pe baza diagramei de clase, nu este foarte complexă. Acest aspect este datorat faptului că sistemul nu urmează șablonul clasic de MVC, având în vedere noțiunile abordate, și etapele necesare de parcurs în modelarea și antrenarea unei rețele neuronale. Structura este adaptată pe aceste etape, proiectul fiind de natură științifică *(Python-Based Data Science Project)*. Astfel, structura simplă a modelului obiectual este rezultatul aplicării etapelor (modulelor) care au realizat procesarea datelor audio, extragerea trăsăturilor muzicale și constuirea rețelei neuronale convoluționale.

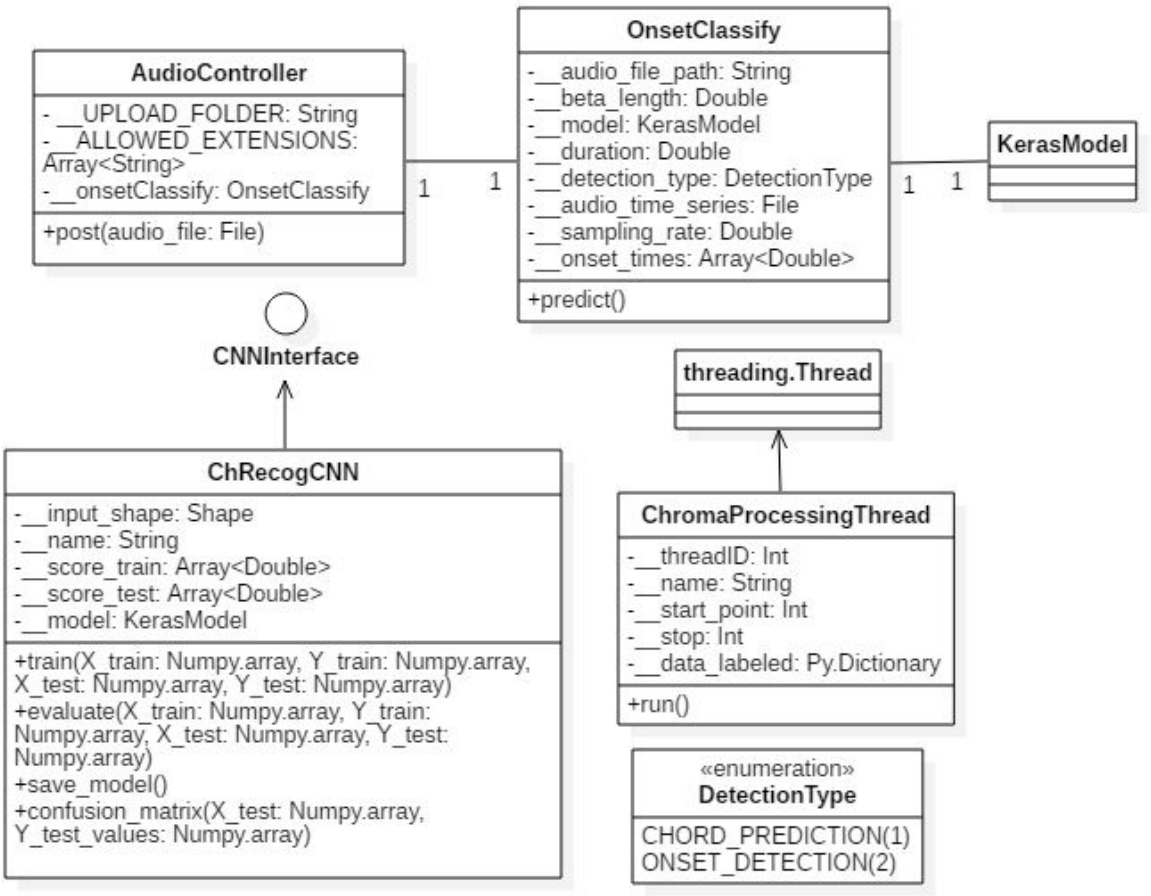

Figura 6.3: Diagrama de clase.

Modulele care compun structura proiectului științific urmează o secvență de pași în mod logic, pornind de la module care se aplică exlusiv setului de date, continuând cu cele care îmbină datele cu modelul neuronal și terminând cu module care sunt destinate doar rețelelor neuronale. Acestea sunt următoarele:

- Intermediate: modul care se ocupă de procesarea datelor și obținerea unor informații și grupări intermediare;
- Processing: la nivelul acestui modul s-a realizat augmentarea datelor și s-a testat pentru prima oară detectația tranzițiilor muzicale;
- Modelling: în acest pas s-a aplicat transformata Q constantă asupra fiecărei înregistrări audio, obținând o listă de chromagrame. Tot aici s-au modelat mai multe arhitecturi convoluționale, ajungând treptat la cea mai bună soluție;
- Model evaluation: modul care utilizează rețeaua antrenată pentru a furniza predicții și pentru a detecta tranzițiile muzicale, în funcție de input-ul dat;
- Utils: conține elementele comune proiectului, ca metricile de calcul, inițializarea sistemului de logging sau constantele utilizate;

- Rest: la nivelul acestui pachet se găsesc endpoint-urile REST API, disponibile la nivelul controller-ului AudioContoller. Acest modul interacționează cu modulul de evaluare.

Cunoscând rolurile îndeplinite de fiecare modul (pachet) în parte, se poate construi o diagramă care să reprezinte dependențele existente între aceste module, dar și componența lor, prin precizarea conținutului (fișiere python ce pot fi rulate independent, clase sau alte obiecte). Figura 6.4 modelează întreg ansamblul enunțat.

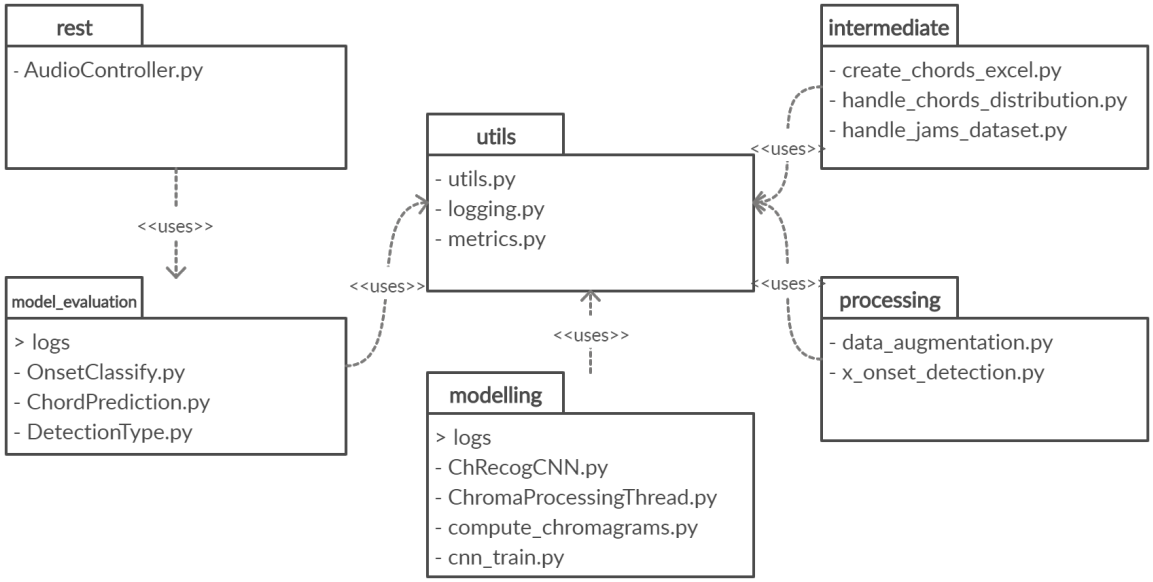

Figura 6.4: Împărțirea sistemului pe module, împreună cu dependențele existente.

Capitolul 7

Abordări existente pentru recunoașterea automată a partiturilor muzicale

Ultimii 20 de ani au adus numeroase abordări în ceea ce privește recunoașterea automată a partiturilor muzicale. De la recunoașterea individuală a instrumentelor muzicale, până la recunoașterea completă a notelor și diferențierea instrumentelor dintr-o piesă completă, s-au încercat diferite metode, în funcție de cantitatea de date disponibilă până în acel moment, dar și de evoluția tehnologiei.

Una din primele lucrări care a stat la baza abordărilor ulterioare pentru o perioadă de timp este lucrarea propusă de către Sheh și Ellis [38]. Metoda are la bază învățare statistică utilizând modele Markov cu stări ascunse. Antrenarea modelului s-a realizat folosind algoritmul EM, tratând etichetele pentru acorduri ca valori ascunse pentru construcția EM. Tot pentru antrenarea modelului au folosit doar secvența de acorduri (*without chord boudaries* - neclasificate, în raport cu intervalele de timp în care se găsește un anumit acord) ca input pentru modelul Markov, aplicând astfel algoritmul Baum-Welch pentru a estima parametrii modelului. Algoritmul garanta că estimările se vor îmbunătății de la o etapă la alta, ajungând într-un optim local, în ceea ce privește setul de parametrii. După determinarea parametrilor, asupra modelului se aplică algoritmul lui Viterbi, pentru a afla cea mai optimă cale(cea mai probabilă secvență de acorduri pentru un semnal audio de intrare).

Rezultatele obținute au fost: 76% pentru precizia segmentării acordurilor din piese și 22% pentru precizia în recunoașterea acordurilor. Performanța scăzută pentru recunoaștere se datorează faptului că datele de antrenare au fost neclasificate, dar și faptului că acestea au fost insuficiente (20 de melodii pentru 147 de acorduri).

Plecând de la această abordare, aproximativ 3 ani mai târziu, Lee și Slaney au enunțat o metodă îmbunătățită [23], având un set de date adnotat și mai numeros, compus din 140 de melodii, obținând o precizie de 93.35% în recunoașterea acordurilor, din cadrul melodiilor testate.

Începând cu anul 2009, odată cu apariția unui set de date compus din piese de pe diverse

albumuri aparținând trupelor Beatles, Zweieck sau Queen, complet adnotate cu acorduri, strategia în rezolvarea acestei probleme s-a schimbat, și foarte multe lucrări au utilizat acest set de date, aplicând diverse metode și comparând rezultate.

O primă lucrare care a abordat mai multe strategii de clasificare, dar și de procesare sonoră, a fost lucrarea de masterat a lui Alessandro Bonvini, din anul universitar 2012-2013, [5]. În ceea ce privește algoritmii de învățare automată, a abordat problema utilizând 3 metode, și anume: regresie logistică (LgR), construirea unui ANN, având un singur strat ascuns *(Single Hidden Multilayer Perceptron (MPL))* și construirea unei rețele de convingeri profunde (DBN).

Bonvini a prezentat rezultatele comparativ, aplicând mai multe strategii de normalizare a datelor, testând la fiecare schimbare toți algoritmii. Se vor prezenta rezultatele obținute prin aplicarea strategiei de normalizare $L-\infty$. În ceea ce privește metricile de evaluare a performanței, a introdus două valori specifice numite Weighted Average Overlap Ratio(WAOR), indicând durata de timp în care segmentele clasificate în mod corect rămân suprapuse celor din realitate, și Segmentation Quality(SQ), sugerând calitatea segmentării acordurilor.

Rezultatele obținute au fost:

- LgR - WAOR: 66.9%, SQ: 74.1%;
- MPL - WAOR: 69.6%, SQ: 74.9%;
- DBN - WAOR: 71.9%, SQ: 76.0 %.

Tot în preajma anului 2012, în cadrul conferinței internaționale de machine learning din Florida, SUA, s-a prezentat o lucrare care avea să schimbe încă odată direcția de a aborda problema recunoașterii automate a acordurilor, prezentând rețelele neuronale convoluționale ca o soluție cu eficiență ridicată, prin comparație cu abordările anterioare, în special prin referire la modelele Markov.

Lucrarea propusă de către Humphrey și Bello, [20], descrie două arhitecturi convoluționale, prezentând construcția strat cu strat și strategia antrenării. O precizare importantă este aceea că, în privința procesării audio și a extragerii de trăsături nu a existat o schimbare, metoda aleasă fiind transformarea Q constantă. Setul de date utilizat conține 475 de înregistrări audio, însumând aproximativ 50000 de acorduri diferite.

Antrenând și testând cele două rețele convoluționale, s-a obținut 77.4%, respectiv 76.8% acuratețe la testare, demonstrând că această abordare performează într-un mod competitiv cu alte abordări considerate consacrate până în acel moment.

Începând din acel moment, majoritatea lucrărilor au propus metode pe baza rețelelor neuronale convoluționale. Cea mai concludentă lucrare analizată, prin raportare la soluția propusă, este cea prezentată de Zhou și Lerch, [45], de la centrul pentru tehnologia muzicii, din cadrul institutului de tehnologie din Georgia, SUA. Lucrarea prezintă construcția a două rețele profunde cu 6 straturi, în două arhitecturii diferite: o arhitectură clasică, cu același număr de neuroni (1024) în fiecare strat, și o arhitectură de tip *bottleneck (cu număr scăzut de neuroni în mijlocul*

rețelei, și în vecinătatea mijlocului). Setul de date constă în 317 piese adunate din discografiile unor trupe celebre, fiecare piesă fiind divizată în peste 1000 de cadre, pentru recunoașterea individuală a acodurilor. Algoritmul propus este capabil să recunoască acordurile majore și minore pentru fiecare notă de tip rădăcină, rezultând un dicționar de etichete pentru acorduri de dimensiunea 24+1, având 24 de acorduri cu etichete corespunzătoare cunoscute, și o etichetă pentru orice alt acord care nu este recunoscut.

Rezultatele obținute sunt prezentate pe ambele arhitecturi, aplicând diferite metode de preprocesare a datelor. Cele mai bune rezultate au fost obținute folosind strategia *spliced filters*. Metrica de evaluare a perfomanței este durata totală a segmentelor cu predicție corectă. Astfel, rezultatele sunt:

- Arhitectura comună - 98.5% la antrenare și 87.6% la testare;
- Arhitectura bottleneck - 93.6% la antrenare și 91.9% la testare.

Se observă o îmbunătățire semnificativă a rezultatelor, construindu-se rețele convoluționale tot mai eficiente, creșterea fiind direct proporțională cu creșterea datelor etichetate corect și în detaliu.

Evoluția studiului în domeniu continuă până în preajma perioadei actuale, fiind prezentată o lucrare foarte importantă acum aproximativ un an de către Nadar, Abeßer și Grollmisch, [29], în cadrul unei conferințe de procesare sonoră, de la Malaga. Lucrarea urmărește extinderea dicționarului de etichete de la 24 la 84 de acorduri, extinzând aria claselor de acorduri care pot fi recunoscute. Dacă până acum se recunoșteau doar acordurile majore și minore ale notelor rădăcină, în această lucrare se dorește recunoașterea și a altor clase, printre care septacorduri sau acorduri diminuate. Setul de date este complex, fiind compus atât din clasicele piese din cadrul discografiilor unor trupe celebre, cât și dintr-un set de date special pentru chitară, creat de Institutul de semantică muzicală, Fraunhofer, din cadrul universității tehnice Ilmenau, Germania.

Modelul convoluțional prezentat este alcătuit din mai multe straturi convoluționale, intercalate de straturi de agregare de tip max pooling, rețeaua având un total de 12 straturi. Rețeaua a fost construită pentru a aborda două stragetii, una care propune recunoașterea pe baza dicționarului extins, dar care nu folosește întreg setul de date (S-84), și una care folosește dicționarul clasic, cu 24 de acorduri (S-24).

Metrica folosită este *f-score*. Astfel, cele mai bune rezultate obținute sunt:

- S-84 - 76%, folosind doar setul de date al institutului Fraunhofer;
- S-24 - 91%, folosind întreg setul de date.

Capitolul 8

Aplicație mobile

În acest capitol se vor prezenta funcționalitățile aplicației mobile și modul de realizare a acestora, din perspectiva cazurilor de utilizare care determină modelul funcțional al sistemului.

8.1 Arhitectura aplicației mobile

Pentru construirea aplicație de mobile am ales modelul de proiectare MVVM (Model-View-ViewModel). Este unul din cele mai bune modele de design care pot fi utilizate pentru a dezvolta o aplicație Android. Acesta este construit din: Model, View și ViewModel [17].

Pachetul Model conține toate clasele care modelează datele prelucrate în aplicație, API-uri dar și obiectele de tip Repository. Astfel, aplicația folosește un obiect WebRepository care implementează funcționalitățile cerute de end-pointurile din WebAPI.

Acesta interacționează cu pachetul de networking, pentru a realiza requesturile dorite, cu ajutorul clientului REST pentru Java și Android, și anume Retrofit. Retrofit face foarte ușor procesul de obținere sau încărcare a unui JSON printr-un serviciu web bazat pe REST. Este necesară doar specificare unui convertor pentru serializarea datelor. Retrofit folosește biblioteca OkHttp pentru solicitările HTTP.

De asemenea, acest pachet interacționează și cu baza de date locală a telefonului. Entitățile sistemului pot fi salvate, la dorința utilizatorului, într-o bază de date relațională, prin librăria Room. Room este o librărie de tip ORM *(Object Relational Mapping)* care transformă automat tabelele SQLite în obiecte Java/Kotlin. Permite validarea SQL în timpul compilării și returnează obiecte observabile de tip RxJava, LiveData sau Flowable.

În cadrul pachetului View se regăsesc structura și aspectele UI cu care utilizatorul poate interacționa. Elementele de view sunt construite prin fragmente și activități.

Activitatea este o componentă crucială a unei aplicații Android. Spre deosebire de alte paradigme de programare în care aplicațiile sunt lansate prin intermediul unei metode principale *main()*, sistemul Android inițializează codul într-o instanță de tip activitate, invocând metode de apelare specifice care corespund unor etape din ciclul de viață a unei activități.

Aplicația conține două activități: MainActivity - activitatea principală, care se lansează la pornirea aplicației, și GuitarMusicPlayerActivity - activitate care se ocupă de procesarea rezultatelor obținute prin trimiterea unei înregistrări audio către sistemul de BE, ce se ocupă cu extragerea trăsăturilor muzicale și furnizarea predicțiilor aferente intervalelor muzicale [42].

Fragmentele sunt porțiuni de interfață din cadrul unei activități. Poate fi considerat o secțiune modulară a unei activități, care are propriul ciclu de viață, fiind gestionat individual de o activitate. Aplicația conține 3 fragmente: HomeFragment - pentru gestiunea înregistrărilor audio salvate local, RecordingFragment - conține un layout care permite înregistrarea unei partituri muzicale, folosind microfonul telefonului, și ChordsFragment - fragment ce conține un grid layout cu reprezentările grafice a tuturor acordurilor recunoscute de către sistem.

Pachetul ViewModel este responsabil de traducerea datelor din Model într-un format corespunzător pentru View. ViewModel și View sunt asociate prin mecanismul de legare a datelelor *(data binding)*, astfel încât orice modificare petrecută la nivel de ViewModel, se poate reflecta automat la nivel de View. Prin urmare, există câte un ViewModel pentru fiecare activitate/fragment existent în aplicație (în afară de MainActivity), prin intermediul cărora se gestionează elementele vizuale din cadrul fragmentelor și a activităților care compun interfața aplicației.

Relațiile de dependență între pachetele modelului sunt reprezentate în Figura 8.1.

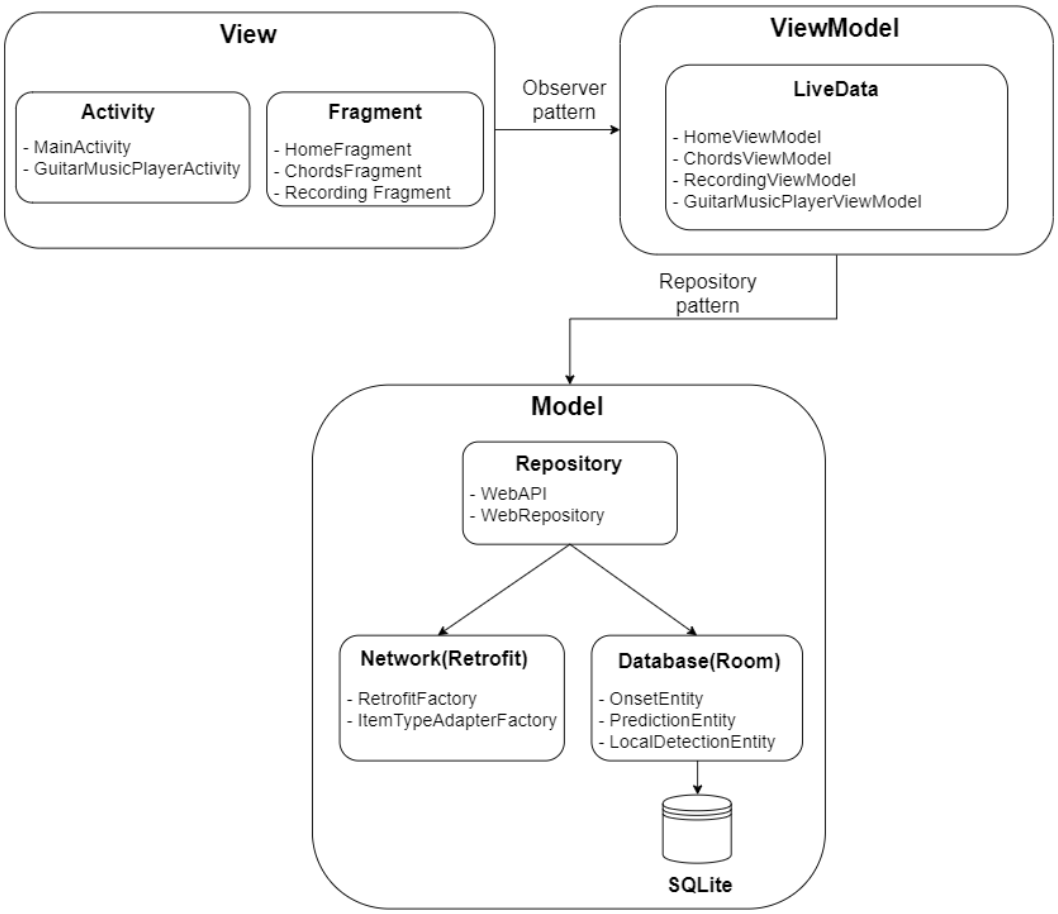

Figura 8.1: Prezentarea modelului de proiectare MVVM al aplicației.

8.2 Funcționalități

Aplicația mobile este construită în jurul ansamblului care se ocupă de recunoașterea automată a partiturilor muzicale acustice, având scopul de a pune în evidență funcționarea sistemului de recunoaștere. De asemenea, punerea în evidență a sistemului trebuie să fie în armonie cu dorințele utilizatorului, astfel încât să rezulte un ansamblu complex, care să satisfacă nevoile utilizatorului, în primul rând prin furnizarea unor rezultate corecte, dar și prin interpretarea acestora într-o manieră simplă și sugestivă, printr-o reprezentare grafică din domeniul muzical.

Funcționalitățile aplicației mobile determină modelul funcțional al sistemului, fiecare caz de utilizare fiind implementat la nivelul unei activități sau a unui fragment al aplicației. Astfel, activitatea principală (MainActivity) este cea care se deschide la pornirea aplicației, si cuprinde 3 fragmente: HomeFragment, RecordingFragment și ChordsFragment.

Fragmentul principal, care este lansat inițial (cazul de utilizare ”Acces pagină principală”), este HomeFragment. Acest fragment are în componență o listă formată din înregistrările muzicale aflate în directorul aplicației mobile (cazul de utilizare ”Gestiune listă fișiere audio wav”). Fiecare element din listă este compus din numele fișierului audio, durata acestuia (minute:secunde), un buton lateral format dintr-o imagine cu 3 linii, prin apăsarea căruia se deschide un submeniu, care oferă posibilitatea de ștergere (cazul de utilizare ”Ștergere fișier”) sau redenumire (cazul de utilizare ”Redenumire fișier”) a fișierului wav. Opțional, elementul poate conține o imagine cu o bifă, ce sugerează faptul că, pentru acel element, informațiile în ceea ce privește recunoașterea automată au fost stocate în baza de date locală a telefonului.

Având aceste informații setate în mod corespunzător pentru fiecare element din listă, se poate realiza cu succes recunoașterea automată a acordurilor muzicale. Pentru a începe acest proces, este nevoie ca utilizatorul să țină apăsat unul din elemente pentru două secunde. Acest gest va determina trimiterea fișierului audio ales către sistemul de recunoaștere, care va realiza operațiile necesare, și va întoarce predicțiile determinte sub forma unui răspuns JSON (cazul de utilizare ”Recunoaște automat acordurile”). Acest gest determină și schimbarea layout-ului. Astfel, peste fragmentul creat, se va crea și afișa activitatea responsabilă de gestiunea rezultatelor sistemului de recunoaștere. Aceată activitate este GuitarMusicPlayerActivity.

Noua activitate instanțiată este responsabilă de afișarea rezultatelor obținute într-un mod cât mai sugestiv. Astfel, activitatea va gestiona un obiect de tip MediaPlayer, care conține fișierul audio de tip wav. Acesta va permite redare conținutului audio. În momentul în care se pornește conținutul audio, în paralel, dar în același timp în mod sincronizat, se vor afișa, cu ajutorul reprezentării prin tabulatură, top 3 acorduri muzicale prezise de către sistemul de recunoaștere automată (cazul de utilizare ”Vizualizare rezultate ale predicției”). Acordurile afișate sunt cele care se potrivesc intervalului muzical din momentul redării (momentul t_i), ceea ce determină o schimbare permanentă a acordurilor afișate, pentru fiecare interval muzical detectat de către sistem (momentul $t_{i+1}, t_{i+2} \dots$).

Fiecare acord muzical reprezentat grafic va fi însoțit de o valoare numerică procentuală, ce

reprezintă probabilitatea asociată predicției de către sistem. Astfel, pentru a avea o vedere mai de ansamblu asupra rezultatelor pentru fiecare interval muzical, s-a ales afișarea și interpretarea primelor 3 predicții.

Tot în această activitate se găsește și opțiunea de a salva/șterge în/din baza de date locală, rezultatele obținute pentru fișierul audio curent. Dacă se alege salvarea rezultatelor pentru acest fișier audio, în viitor, când se va dori din nou o vizualizare a predicțiilor corespunzătoare, acestea vor fi luate și afișate din baza de date locală, fiind disponibile mult mai rapid, întrucât procesul complex de recunoaștere și furnizare a răspunsului din partea sistemului de recunoaștere nu mai are loc. Sumarizarea funcționalităților enunțate mai sus este redată prin Figura 8.2, care prezintă cele două layere descrise, HomeFragment și GuitarMusicPlayerActivity.

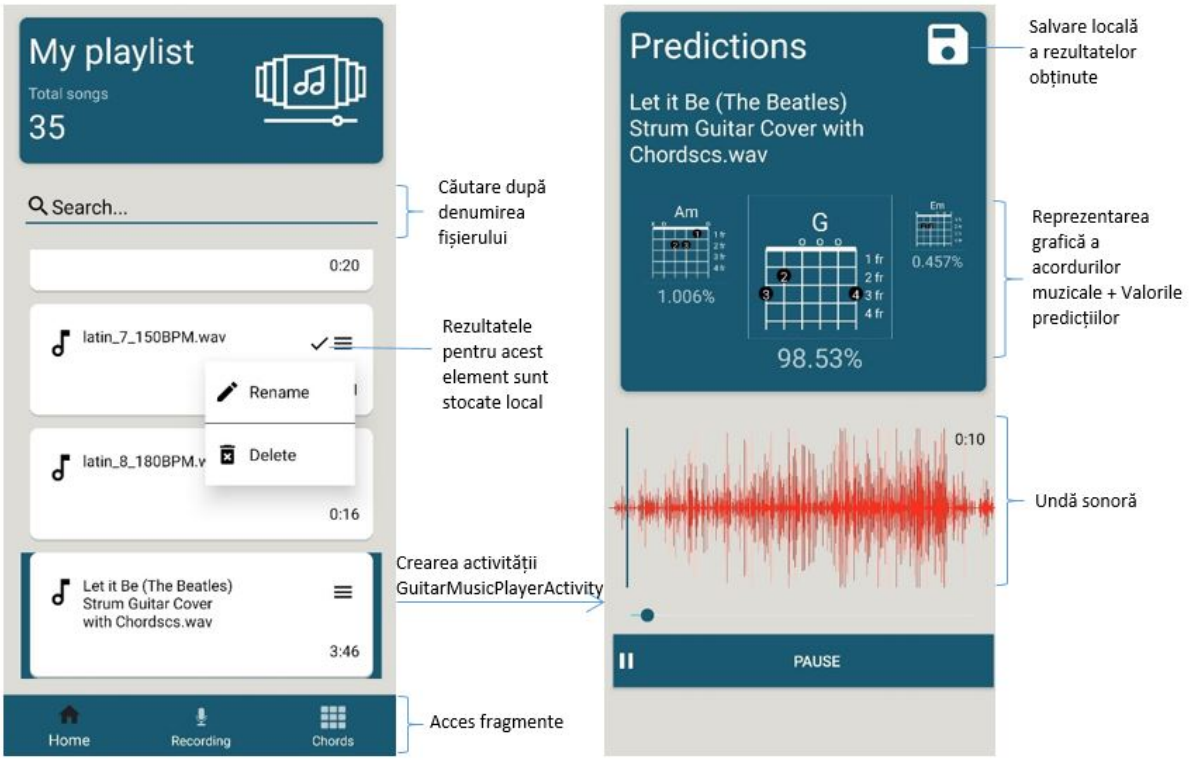

Figura 8.2: Imaginea din stânga prezintă elementele fragmentului HomeFragment. Prin ținerea apăsată a unuia din elemente, se instanțează și apoi se afișează activitatea GuitarMusicPlayerActivity (imaginea din dreapta), responsabilă de procesarea și afișarea rezultatelor furnizate de sistemul de recunoaștere automată.

Cele două elemente de View prezentate mai sus sunt cele mai importante, fiind în legătură strânsă cu sistemul de recunoaștere automată a partiturilor muzicale. Pe langă ele, mai există în aplicație două fragmente auxiliare, anume RecordingFragment și ChordsFragment.

Fragmentul de recording permite utilizatorului să folosească microfonul telefonului (sau un microfon extern atașat) pentru a înregistra o partitură muzicală (cazul de utilizare ”Înregistrare audio”). Odată ce acesta finalizează înregistrarea, poate să salveze conținutul sub forma unui fișier wav, care va fi stocat în directorul aplicației, sau să șteargă conținutul și să reîncerce re-

alizarea unei înregistrări (cazurile de utilizare ”Salvare înregistrare” și ”Ascultă înregistrarea”). Ulterior, poate folosi înregistrea pentru a descoperi acordurile muzicale, prin trimiterea acesteia către sistemul de recunoaștere automată.

Ultimul fragment, ChordsFragment, are un rol didactic în cadrul aplicației. Fragmentul conține o listă cu elemente dispuse sub forma unei grile *(grid layout)*, fiecare element reprezentând unul din cele 24 de acorduri recunoscute de către sistem. Astfel, fiecare element din listă prezintă numele acordului muzical, împreună cu reprezentarea lui grafică (desenul), fiind folosită notația prin tabulatură (cazul de utilizare ”Vizualizare listă acorduri muzicale”). Prin selectarea unui element din listă, se afișează o fereastră de tip popup, care prezintă acordul, și un buton de start. Prin apăsarea butonului, dispozitivul mobil redă un sunet de aproximativ 2 secunde, ce reprezintă acordul muzical selectat, interpretat la o chitară acustică (cazul de utilizare ”Ascultă acord”). Figura 8.3 ilustrează cele două fragmente descrise mai sus.

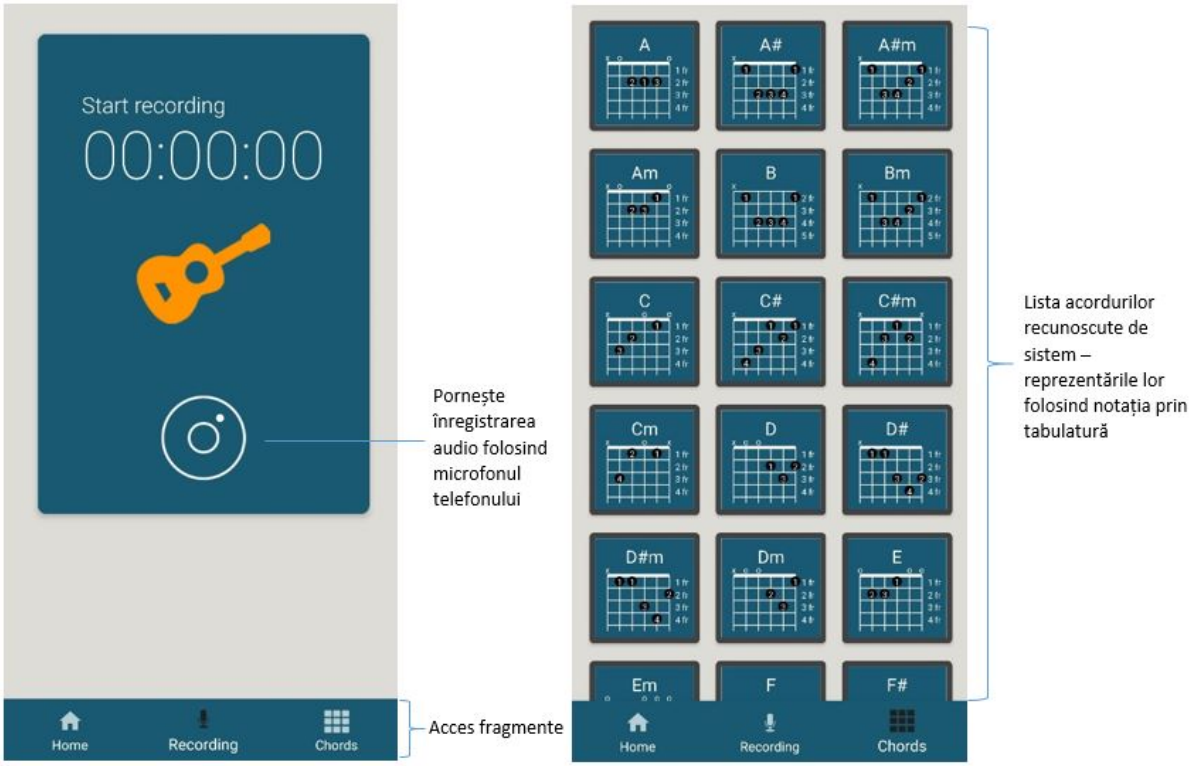

Figura 8.3: Imaginea din stânga reprezintă fragmentul RecordingFragment, responsabil de înregistrarea audio, în timp ce imaginea din dreapta prezintă lista acordurilor muzicale recunoscute de către sistem (folosind notația prin tabulatură).

8.3 Proiectarea interfeței grafice

Atât interfața grafică, cât și aspectul aplicației mobile, au fost proiectate folosind fișiere XML. Folosind vocabularul XML oferit de Android, se pot proiecta rapid diverse layout-uri pentru interfața grafică, împreună cu elementele corespunzătoare, în același mod în care se creează pagini Web în HTML - cu o serie de elemente imbricate.

Fiecare fișier de layout trebuie să conțină exact un element rădăcină, care trebuie să fie un obiect View sau ViewGroup. După ce se definește obiectul rădăcină, se pot adăuga obiecte sau elemente suplimentare ca elemente ”copii”, în raport cu rădăcina, pentru a construi treptat o ierarhie de vizualizare care să definească layout-ul. În ceea ce privește rădăcina, cele mai frecvente alegeri sunt layout-urile următoare: LinearLayout, ConstraintLayout sau RelativeLayout. De asemenea, la nivel de elemente componente, se utilizează frecvent elemente ca: TextView, Button, ImageView sau ListView.

Fiecare obiect View acceptă propria varietate de atribute. Unele atribute sunt specifice unui obiect (de exemplu TextView, care acceptă atributul textSize), dar pot exista și atribute comune (cum ar fi atributul id). De asemenea, există atribute considerate ”parametrii de layout”, acestea având rolul de a descrie anumite orientări de layout ale acelui obiect, valori care se bazează și pe cele setate de obiectul părinte.

În ceea ce privește fișierele xml care definesc layout-ul aplicației (4 fișiere, 3 pentru fragmente și unul pentru o activitate), acestea au folosit în ierarhizare mai multe tipuri de componente, pentru a da un aspect plăcut și modern aplicației. Se consideră ca exemplu, cel mai complex fragment, și anume HomeFragment. Acesta este format dintr-un LinearLayout (ca rădăcină), având în componență un view de tip CardView (compus dintr-o ierarhie de LinearLayout-uri, TextView-uri și un ImageView), un EditText (pentru realizarea filtrării), și un layout de tip include, care determină integrarea unei liste din exterior (dintr-un alt fișier xml). Secțiunea de cod de mai jos prezintă o sumarizare a componentelor enunțate, și a modului de compunere a acestui layout.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#DDDCD7"
    android:orientation="vertical">
    <androidx.cardview.widget.CardView
        android:layout_width="match_parent"
        android:layout_height="150dp"
        ...
        <LinearLayout ...
            <LinearLayout ...
                <TextView ...
                    android:text="My playlist"
                    android:textSize="32dp" />
```

```
                ...
            </LinearLayout>
            <ImageView ... />
        </LinearLayout>
</androidx.cardview.widget.CardView>
<EditText ...
    android:hint="Search..."
    .../>
<include
    layout="@layout/listview"
    ... />
</LinearLayout>
```

Acest stil de realizarea a unui layout este similar și pentru construcția fragmentelor rămase. Componentele utilizate sunt aceleași, diferă doar ordinea lor, dar și atributele utilizate. Dacă în ceea ce privește fragmentele, domină layout-ul de tip LinearLayout (atât ca rădăcină, cât și ca element în structura ierarhică), situația este puțin diferită pentru activitate. Activitatea GuitarMusicPlayerActivity este formată dintr-un ConstraintLayout (ca rădăcină) ce are în componență elemente de tip LinearLayout, TextView, ImageView și CardView. Spre deosebire de LinearLayout, care ierarhizează componentele în mod liniar, după orientarea aleasă (orizontal sau vertical), pentru fiecare element din cadrul unui ConstraintLayout trebuie precizate constrângerii în relație cu componentele vecine (atribute de tipul layout_constraint). Modul de definire a unui layer de tip Constraint se poate observa în secțiunea de cod de mai jos.

```
<androidx.constraintlayout.widget.ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#DDDCD7"
    tools:context=".ui.music_player.GuitarMusicPlayerActivity">

    <androidx.cardview.widget.CardView
        android:id="@+id/cardView2"
        android:layout_width="match_parent"
        android:layout_height="360dp"
        .../>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">
        ...
    </LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Capitolul 9

Concluzii

9.1 Rezumat

În această lucrare s-au prezentat abordări pentru problema recunoașterii automate a partiturilor muzicale. S-a observat că rezolvarea problemei este dependentă de parcurgerea a două etape principale. Prima etapă se referă la metode de procesare ale semnalului sonor, și obținerea unei reprezentări semnificative, sub forma unei imagini. S-au prezentat aspectele teoretice și practice a două metode, trasformata Fourier pe termen scurt și transformata Q constantă. În ceea ce privește reprezentarea semnalului, s-a ales reprezentarea prin chromagramă, prezentându-se tehnici de compunere a acesteia.

Etapa a doua este reprezentată de introducerea algoritmilor de învățare automată, cu focus pe rețele neuronale profunde. Rolul acestora este de a crea un model neuronal bazat pe reprezentările înregistrărilor audio, model care va fi folosit în clasificarea acordurilor detectate.

Aceste două etape au fost aplicate asupra unui set de date format din înregistrări audio de tip wav (fișiere audio originale și augmentate). Astfel, pentru procesarea sunetului s-a utilizat transformata Q constantă, pe baza căreia s-a obținut chromagrama fiecărui fișier audio. Cu lista de chromagrame (matrici de caracteristici) construită, pentru pasul următor, s-a construit, antrenat și evaluat o rețea neuronală convoluțională. Prin analiza rezultatelor obținute pentru metricile de calcul și prin analiza graficelor, s-a observat că acest model furnizează rezultate excelente, fiind o alegere foarte bună pentru acest domeniu. Modelul neuronal a fost combinat cu un algoritm de detecție a tranzițiilor muzicale, pentru a face posibilă recunoașterea partiturilor pe intervalele muzicale observate, obținând astfel rezultate complete și corecte pentru întreg conținutul muzical.

Pe baza sistemului de recunoaștere construit, s-a realizat și o aplicație mobile destinată utilizatorilor care îndrăgesc muzica acustică la chitară. Aplicația permite gestiunea unui playlist de piese acustice, care pot fi trimise, la dorința utilizatorului, către sistemul de recunoaștere, care va întoarce rezultatele prezise, aplicația având rolul de a reprezenta grafic aceste rezultate, utilizând notația prin tabulatură. Aplicația mai permite înregistrarea unei piese, folosind

microfonul telefonului. De asemenea, mai pune la dispoziția utilizatorului o listă cu toate reprezentările sub formă de tabulatură recunoscute de către sistem, având un scop didactic pentru cei dornici să învețe modul de realizare la chitară a acordurilor muzicale.

9.2 Îmbunătățiri viitoare

Calitatea clasificării obținute prin tehnicile de învățare automată este strict legată de calitatea informațiilor muzicale extrase. Din acest motiv, se poate investiga obținerea și folosirea unei reprezentări mai bune decât chromagrama, prin analiza unor metode diferite de procesare a semnalului sonor. Astfel, s-ar putea obține o îmbunătățire a trăsăturilor muzicale extrase din înregistrările audio acustice.

În ceea ce privește metodele de învățare automată utilizate, ar putea fi luată în considerare o abordare care să folosească un alt algoritm de învățare supervizată. S-a observat prin realizarea cercetărilor conexe că rețele neuronale de convingeri profunde pot furnizare rezultate interesante, ceea ce face ca această arhitectură să merite o analiză mai amplă, dar și o implemetare pentru a îi dovedi calitățile și utilitatea. De asemenea, un model neuronal modern care nu este abordat în această lucrare este cel al rețelelor neuronale recurente, având și ele un rol important în prelucrarea, modelarea și clasificare imaginilor.

O altă analiză înspre îmbunătățirea sistemului poate fi făcută pentru algoritmul de detecție a tranzițiior muzicale. Detecția tranzițiilor este o arie separată de cea a recunoașterii partiturilor muzicale, având numeroase abordări la rândul ei. Se pot detecta tranzițiile muzicale inclusiv prin creearea unei rețele neuronale convoluționale, întrucât există seturi de date adnotate corespunzător, cu ajutorul cărora se poate realiza un model neuronal specializat doar pe obținerea intervalelor muzicale, intervale care să fie folosite mai departe pentru alți algoritmi sau metode.

Deși rezultatele obținute prin metodele alese pentru implementare sunt foarte bune, cu siguranță există loc de îmbunătățiri, mai ales dacă se realizează și o îmbogățire a setului de date. Recunoașterea automată a partiturilor muzicale este o temă cu multiple abordări, fiecare aducând un element de noutate domeniului. Este clar că evoluția tehnologiei și a puterii de calcul în ceea ce privește învățarea automată se va reflecta și asupra acestui domeniu, iar în viitor vor exista modele de recunoaștere automată tot mai exacte, fiind capabile să acopere arii tot mai mari din sfera muzicii acustice contemporane, și nu numai.

Bibliografie

[1] *Analog to digital converion process*. Accesat în 2020-03-05. July 2014. URL: https://victorimpmooc.wordpress.com.

[2] Juan P Bello and Jeremy Pickens. *A robust mid-level representation for harmonic content in music signals.* In prococeedings of ISMIR, pag. 304-311, 2005.

[3] Johan Bjorck, Carla Gomes, Bart Selman, and Kilian Q. Weinberger. *Understanding Batch Normalization*. In: *Cornell University* (2018).

[4] Sebastian Bock, Florian Krebs, and Markus Schedl. *Evaluating the online capabilities of Onset detection methods*. In: *Department of Computational Perception* (2012).

[5] Alessandro Bonvini. *Automatic chord recognition using Deep Learning techniques*. In: *Journal: POLITECNICO DI MILANO Facoltà di Ingegneria dell'Informazione Corso di Laurea Magistrale in Ingegneria e Design del suono Dipartimento di Elettronica e Informazione* (2012-2013).

[6] R. N. Bracewell. *The Fourier Transform and Its Applications (ed. 3rd)*. McGraw-Hill, ISBN-13: 9781124130941, 2000.

[7] Judith C Brown. *Calculation of a constant q spectral transform*. The Journal of the Acoustical Society of America, 1991.

[8] *C-sharp note*. Accesat în 2020-05-04. URL: https://www.basicmusictheory.com/c-sharp-note.

[9] *Ce este o tabulatură?* Accesat în 2020-05-04. URL: https://www.tabulaturi.ro/lectii/ce-este-o-tabulatura.

[10] *CS231n: Convolutional Neural Networks for Visual Recognition, by Stanford University*. Accesat în 2020-04-06. URL: https://cs231n.github.io/convolutional-networks/overview.

[11] Dan-Marius Dobrea. *Tehnici de inteligență computațională. Aplicații în electronică și biomedicină*. Editura Performantica (editură acreditată CNCSIS), Iaşi, România, ISBN 978-606-685-546-4, 2017.

[12] *Easy Tips for Reading Guitar Chord Charts*. Accesat în 2020-05-04. URL: https://takelessons.com/blog/guitar-chord-charts-z01/.

[13] *Fourier Transform Visualization, based on video presentation. Experiment using project*. URL: https://prajwalsouza.github.io/Experiments/Fourier-Transform-Visualization.html.

[14] Munteanu Gabriela. *Educație muzicală, Sinteze și corelări interdisciplinare*. Editura Artrom, 2001.

[15] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow*. O'Reilly Media, ISBN-13: 9781492032649, 2019.

[16] *Google Colab*. Accesat în 2020-04-26. URL: https://www.tutorialspoint.com/google_colab/what_is_google_colab.htm.

[17] *Guide to app architecture*. Accesat în 2020-05-12. URL: https://developer.android.com/jetpack/docs/guide.

[18] *Guide to PyCharm for Windows and MacOS*. Accesat în 2020-04-26. URL: https://kite.com/blog/python/pycharm/.

[19] Hartmann and William Morris. *Signals, Sound, and Sensation*. Springer. pp. 145, 284, 287. ISBN 978-1-56396-283-7.

[20] Eric J. Humphrey and Juan P. Bello. *Rethinking Automatic Chord Recognition with Convolutional Neural Networks*. In: *Music and Audio Research Lab (MARL) New York University* (2012 11th International Conference on Machine Learning and Applications).

[21] Daniel Jurafsky and James H. Martin. *Speech and Language Processing, An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Draft of October 16, ISBN-13: 9789332518414, 2019.

[22] *Kotlin overview*. Accesat în 2020-05-02. URL: https://developer.android.com/kotlin/overview.

[23] Kyogu Lee and Malcolm Slaney. *Automatic Chord Recognition from Audio Using an HMM with Supervised Learning*. In: *ISMIR 2006, 7th International Conference on Music Information Retrieval* (2006).

[24] *LibROSA*. Accesat în 2020-04-06. URL: https://librosa.github.io/librosa/index.html.

[25] *Machine Learning Crash Course*. Accesat în 2020-05-02. URL: https://developers.google.com/machine-learning/crash-course.

[26] *MathWorks-Convolutional Neural Network*. Accesat în 2020-04-08. URL: http://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html.

[27] *Meet Android Studio*. Accesat în 2020-05-02. URL: https://developer.android.com/studio/intro.

[28] Elias Mistler. *Generating Guitar Tablatures with Neural Networks*. In: *School of Informatics, University of Edinburgh* (2017).

[29] Christon-Ragavan Nadar, Jakob Abeßer, and Sascha Grollmisch. *Towards CNN-based Acoustic Modeling of Seventh Chords for Automatic Chord Recognition*. In: *Sound and Music Computing Conference (SMC), Malaga* (2019).

[30] *Notiuni esentiale despre acorduri la o chitara acustica*. Accesat în 2020-05-04. URL: https://promusicprod.ro/notiuni-esentiale-despre-acorduri-la-o-chitara-clasica/.

[31] *NumPy*. Accesat în 2020-05-02. URL: https://numpy.org/.

[32] *Onset Detection*. Accesat în 2020-04-28. URL: https://musicinformationretrieval.com/onset_detection.html.

[33] *Pandas 1.0.3*. Accesat în 2020-05-02. URL: https://pypi.org/project/pandas/.

[34] J. Pauwels, Q.Xi, R.Bittner, J.P.Bello, and X.Ye. *Guitarset, A Dataset for Guitar Transcription*. In: *19th International Society for Music Information Retrieval Conference, Paris, France* (2018).

[35] Preeti Rao. *Audio Signal Processing, Chapter in Speech, Audio, Image and Biomedical Signal Processing using Neural Networks*. In: *Bhanu Prasad and S. R. Mahadeva Prasanna, Springer-Verlag* (2007).

[36] *ResearchGate - Example of the probabilistic parameter of a hidden Markov model*. Accesat în 2020-04-05. URL: https://www.researchgate.net/.

[37] *Robotics, Vision and Control (ROVIS) Laboratory*. Accesat în 2020-04-08. URL: http://www.rovislab.com/courses/ml/.

[38] Alexander Sheh and Daniel P.W. Ellis. *Chord Segmentation and Recognition using EM-Trained Hidden Markov Models*. In: *International Symposium on Music Information Retrieval* (2003).

[39] *The Music Studio*. Accesat în 2020-03-29. URL: https://www.themusicstudio.ca/blog/2017/11/909/.

[40] Cornelia Novac Ududec. *Ingineria sistemelor de programare*. Editura Alma Mater Bacău 2011.

[41] *Ultimate Guitar*. Accesat în 2020-05-04. URL: https://www.ultimate-guitar.com/.

[42] *Understand the Activity Lifecycle*. Accesat în 2020-05-12. URL: https://developer.android.com/guide/components/activities/activity-lifecycle.

[43] *What is Python? Executive Summary*. Accesat în 2020-04-26. URL: https://www.python.org/doc/essays/blurb/.

[44] *Working with CNN 2D Convolutions in Keras*. Accesat în 2020-04-06. URL: https://missinglink.ai/guides/keras/keras-conv2d-working-cnn-2d-convolutions-keras/.

[45] Xinquan Zhou and Alexander Lerch. *Chord detection using Deep Learning*. In: *ISMIR 2015, International Conference on Music Information Retrieval* (2015).