UNIVERSITATEA POLITEHNICA DIN BUCUREȘTI
FACULTTEA DE ELECTRONICĂ, TELECOMUNICAȚII ȘI TEHNOLOGIA INFORMAȚIEI

SISTEM DE RECUNOAȘTERE ȘI CLASIFICARE A PLÂNSETELOR NOU-NĂSCUȚILOR FOLOSIND DEEP LEARNING

PROIECT DE DIPLOMĂ

prezentat ca cerință parțială pentru obținerea titlului de *Inginer* în domeniul *Calculatoare și Tehnologia Informației*

Programul de studii *Ingineria Informației*

Profesor coordonator
Ș.L. Dr. Ing. Horia Cucu

Student
Radu-Ștefan DELEANU

București
2020

Universitatea “Politehnica” din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Departamentul **EAII**

Anexa 1

TEMA PROIECTULUI DE DIPLOMĂ

a studentului **DELEANU Gh. Radu-Ştefan , 441A**

1. Titlul temei: Sistem de recunoaștere și clasificare a plânsetelor nou-născuţilor folosind deep learning

2. Descrierea contribuţiei originale a studentului (în afara părţii de documentare) și specificații de proiectare:
O serie de studii ştiinţifice au arătat că nou-născuţii (0-3 luni) plâng diferit în funcţie de nevoile pe care le au: foame, somn, discomfort fizic, durere, etc.
Acest proiect propune realizarea unui sistem de recunoaștere și clasificare a plânsetelor nou-născuților în funție de nevoile acestora, folosind fragmente audio care provin dintr-o bază de date pre-existentă. Implementarea acestui sistem se bazează pe tehnici moderne de machine learning (deep learning - reţele neuronale recurente, convoluţionale, etc.) şi, eventual, utilizarea formei de undă la intrarea sistemului.
După ce s-a realizat împărțirea bazei de date în loturi de antrenare, validare și test, se va folosi programul openSMILE pentru extracția de trăsături din fișierele wav, folosind un fișier de configurare corespunzător setului de trăsături ComParE. În urma procesului de extracție rezultă fișiere de tip CSV. Astfel se obține o uniformizare a datelor de intrare, toate exemplele având un număr fix de parametrii, făcând posibilă antrenarea unui clasificator de tip MLP (Multilayer Perception). Se va experimenta cu diferite configurații arhitecturale și valori pentru hiper-parametri, în scopul obținerii unei performațe cât mai bune a modelului.
Ulterior se vor încerca și alte abordări folosind Rețele Neurale Convoluționale (ConvNets) și Rețele Neuronale Residuale (ResNet), eliminandus-se constrângerea impusă de o rețea de tip MLP cu privire la dimensiunea fixă a datelor de intrare. Astfel se vor putea antrena modele folosind direct fișierele wav din baza de date.

3. Resurse folosite la dezvoltarea proiectului:
Python, Framework Deep Learning - PyTorch, Extractor de trasaturi - openSMILE, Baza de Date

4. Proiectul se bazează pe cunoştinţe dobîndite în principal la următoarele 3-4 discipline:
RFIA, SDA, SS, IOM

5. Proprietatea intelectuală asupra proiectului aparţine: U.P.B.

6. Data înregistrării temei: 2020-02-28 20:37:37

Conducător(i) lucrare,
Conf. dr. ing. Horia CUCU

Student,

Director departament,
Prof. dr. ing Sever PAȘCA

Decan,
Prof. dr. ing. Mihnea UDREA

Cod Validare: **8082343d90**

Declarație de onestitate academică

Prin prezenta declar, că lucrarea cu titlul „SISTEM DE RECUNOAȘTERE ȘI CLASIFICARE A PLÂNSETELOR NOU-NĂSCUȚILOR FOLOSIND DEEP LEARNING"

prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității "Politehnica" din București ca cerință parțială pentru obținerea titlului de *Inginer* în domeniul ~~Inginerie Electronică și Telecomunicații~~/Calculatoare și Tehnologia Informației, programul de studii *Ingineria Informației* este scrisă de mine și nu a mai fost prezentată niciodată la o facultate sau instituție de învățământ superior din țară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referințe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și fac referință la sursă. Reformularea în cuvinte proprii a textelor scrise de către alți autori face referință la sursă. Înțeleg că plagiatul constituie infracțiune și se sancționează conform legilor în vigoare.

Declar că toate rezultatele simulărilor, experimentelor și măsurătorilor pe care le prezint ca fiind făcute de mine, precum și metodele prin care au fost obținute, sunt reale și provin din respectivele simulări, experimente și măsurători. Înțeleg că falsificarea datelor și rezultatelor constituie fraudă și se sancționează conform regulamentelor în vigoare.

București, Septembrie 2020

Absolvent RADU-ȘTEFAN DELEANU

CUPRINS

LISTĂ DE FIGURI

LISTĂ DE TABELE

LISTĂ DE ACRONIME

API - Application Programmable Interface

CPU - Central Processing Unit

CSV - Comma Separated Values

CUDA - Compute Unified Device Architecture

GAN - Gradient Accelerat Nesterov

GPU - Graphics Processing Unit

MFCC - Mel Frequency Cepstral Coefficient

RAM - Random Access Memory

ReLU - Rectified Linear Unit

SGD - Stochastic Gradient Descent

SVM – Support Vector Machines

WAV - Waveform Audio File Format

CAPITOLUL 1

INTRODUCERE

1.1 MOTIVAȚIE

Dorința de a lucra la un proiect de cercetare în spațiul inteligenței artificiale și aspirația de a avea o contribuție utilă într-un domeniu emergent sunt prinicipalele motive pentru care s-a realizat această teză.

Cercetările recente din domeniul neonatologiei au consolidat teoria conform căreia plânsetele bebelușilor care nu au împlinit vârsta de 3 luni conțin informații relevante pentru necesitățile acestora. După vârsta de trei luni, nu se pot mai extrage informații cu caracter definitoriu pentru starea bebelușului din plânset, deoarece aparatul vocal al acestora se dezvoltă semnificativ pentru a produce sunete mai complexe, însă aceste prime câteva luni de viață sunt foarte importante și o mai bună înțelegere a nevoilor ar ușura semnificativ procesul de îngrijire. [1]

Prinicipalii beneficiari ai rezultatelor obținute în urma cercetărilor prezentate în această lucrare sunt părinții tineri, deoarece un sistem automat capabil să determine nevoia pe care o resimte un bebeluș ar fi un real ajutor. Pentru părinții care aduc pe lume primul copil, întelegerea nevoilor pe care le au nou-nascuții este destul de dificilă, fiind o exeprientă pentru care nu se pot pregăti anterior nașterii. În ceea ce privește beblușul, una dintre puținele forme de comunicare pe care le poate manifesta este plânsul, și astfel abilitatea de a recunoaște cu succes ce nevoie se află în spatele plânsetului este foarte valoroasă pentru un părinte la început de drum. Câteva dintre motivele pentru care un bebeluș plânge sunt: foamea, oboseala, durerea, eructația sau colicile. De

multe ori, părinții nu pot determina care este motivul din spatele manifestației vocale a bebelușului și astfel nu știu cum să procedeze pentru a remedeia situația și a calma nou-născutul.

S-a observat ca neonatologiștii pot distinge între câteva tipuri de plânset, însă expertiza acestora este în principal bazată pe experiență acumulată în mult timp. De asemenea, interpretările acestora asupra plânsetelor pot fi subiective. O serie de factori precum construcția fizică, vârsta, sexul sau greutatea bebelușului contribuie de asemenea la compoziția plânsetului, făcând aproape imposibilă o standardizare în ceea ce privește recunoașterea nevoii. Aceste aspecte atestă necesitatea dezvoltării unui sistem automat, capabil să generalizeze și să discearnă nevoile ascunse în spatele plânsetului, invariant la particularitâțile anatomice ale nou-născutului.

1.2 OBIECTIVE

Considerând importanța pe care o prezintă subiectul tratat în cadrul medicinei moderne, această lucrare este menită să consolideze cercetările anterioare bazate pe tehnici de învățare automată în domeniul clasificării plânsetelor de bebeluși. Acest proiect își propune o abordare diferită, bazată pe tehnici de învațare profundă, având în vedere următoarele obiective:

a. Generarea unui baze de date formată din trăsături extrase din baza de date SPLANN
b. Proiectarea și dezvoltarea unui sistem de clasificare folosind o arhitectură de rețea neurală profundă personalizată și observarea performanțelor acesteia în funcție de variația hiperparametrilor.
c. Implementarea și testarea unor arhitecturi de rețele neurale deja existente pe baza de date generată anterior și studiul performanțelor acestora comparativ cu cele obținute de rețeaua personalizată.

1.3 STRUCTURA LUCRĂRII

Capitolul 1 conține informații în legatură cu motivația, obiectivele și structura lucrării.

Capitolul 2 conține informații în legatură cu starea artei în recunoaștere de plânset de bebeluș. Astfel, se vor prezenta bazele de date folosite în experimentele aferente tezei prezente, lucrări anterioare de cercetare pe aceeași temă și rezultatele obținute în cadrul acestora, cât și indicii de performanța ce pot fi folosiți în evaluarea clasificatorilor.

Capitolul 3 descrie noțiuni teoretice despre metoldele folosite cât și despre conceptele matematice care stau la baza sistemelor de clasificare. Se va prezenta pe scurt setul de trăsături ComParE folosit in cadrul proiectului iar apoi se vor detalia elementele constituente, structura și compoziția unei rețele neurale artificiale.

Capitolul 4 prezintă câteva aspecte despre tehnologiile folosite in dezvoltarea lucrării și în partea exerimentală. Astfel, se vor prezenta limbajul de programare Python, librăriile Pytorch și Scikit-learn, platforma de dezvoltare Google Colab, extractorul de trăsături openSMILE și platforma de calcul paralel CUDA. Descrierea fiecărei tehnolgii este urmată de o menținue a rolului său în realizarea proiectului.

Capitolul 5 expune la modul general structura programelor care implementează experimentele, menționând rolul ficărui modul sau bloc funcțional. Vor fi detaliate apoi tipurile de rețele folosite în cadrul experimentelor, prezentându-se atât aspectele formale, cât și informațiile pe care le revelă fiecare arhitectură în parte despre subiect.

Capitolul 6 aduce detalii despre implementarea propriu-zisă a sistemelor de clasificare și a experimentelor realizate în cadrul proiectului. Se vor parcurge etapele de extragere de trăsături, încărcarea datelor, preprocesare și formatare, definirea modelului, antrenare și în final evaluarea performanțelor. Se vor prezenta pe scurt și modulele responsabile cu salvarea și încărcarea modelelor precum și celulele utile în reprezentarea rezultatelor.

Capitolul 7 este dedicat părții experimentale a proiectului. În acest capitol se va prezenta configurația experimentală, aducându-se detalii despre setul de date și împarțirea acestuia cât și despre resursele hardware și software folosite. Ulterior se vor prezenta rezultatele obținute de fiecare tip de arhitectură în parte.

Capitolul 8 conține observații și concluzii referitoare la rezultatele experimentelor. De asemenea, se vor expune contribuțiile personale și posibile dezvoltări ulterioare ale proiectului.

CAPITOLUL 2

STAREA ARTEI ÎN RECUNOAȘTERE DE PLÂNSET DE BEBELUȘ

2.1 BAZA DE DATE DUNSTAN

Originea acestei baze de date poate fi atribuită observațiilor mezzo-sopranei australiene Priscilla Dunstan. Aceasta a descoperit la o vârstă fragedă abilitatea de a distinge, interpreta și reproduce o suită de sunete complexe. Ulterior aceasta a extins abilitatea sa în sfera plânsetelor nou-născuților, identificând anumite tipare și similarități constituente pentru cinci categorii de sunete. Observațiile sale erau relevante la mai mulți indivizi, indiferent de rasă sau sex.

Pricilla Dunstan a teoretizat că bebelușii cu vârste cuprinse între 0 și 3 luni prezintă anumite reflexe sonore, care aparent urmează un tipar. Aceste refelxe sonore consituie un plânset incipient, relativ la nevoia pe care cel mic o manifestă. Dacă disconfortul resimțit nu este ameliorat în timp util, bebelușul scoate un plânset isteric. [1]

Din dorința de validare științifică a teoriei, a fost creată baza de date Dunstan. Aceasta a fost etichetată de medici neonatologi și medici pediatrii în cinci tipuri diferite de plânset: foame, dureri de burtă, somn, disconfort și eructație. Pentru început, studiile au inclus în analiză și imagini ale nou-născuților în timpul plânsului, din considerente de completare a contextului nevoii pe care aceștia o prezentau. Baza de date Dunstan conține proble audio de la un număr total de 37 de bebeluși, care au generat date pentru mai multe clase după cum urmează:

- plânsete reprezentative pentru foame: au generat probe 23 de bebeluși;

- plânsete reprezentative pentru dureri de burtă: au generat probe 10 de bebeluși;

- plânsete reprezentative pentru somn: au generat probe 16 de bebeluși;

- plânsete reprezentative pentru disconfort: au generat probe 12 de bebeluși;

- plânsete reprezentative pentru eructație: au generat probe 12 de bebeluși;

Această baza de date constituie un punct de plecare în cercetarea legată de recunoașterea și clasificarea plânsetelor de nou-născuților din ultimii ani.

2.2 BAZA DE DATE SPLANN

Avansarea cercetării în domeniu a determinat nevoia unei baze de date mai cuprinzătoare și mai voluminoasă ca numar de probe, pentru a reuși îndeplinirea sarcinii de clasificare cu mai multă precizie. Această direcție s-a materializat sub forma proiectului Automatic Infant Crying Recognition System (SPLANN) [2], realizat de către compaia de dezvoltare software SOFTWIN în parteneriat Facultatea de Electronică, Telecomunicații și Tehnologia Informației, Universitatea Politehnica București și Spitalul Clinic de Urgență "Sfântul Pantelimon" din București [3]. În urma acestui proiect au fost colectate și etichetate date din lumea reală de la un numar considerabil de subiecți, extinzând semnificativ acoperirea bazei de date Dunstan.

Colectarea de date propriu-zisă s-a realizat în principal în incinta maternității, folosindu-se echipament specilaziat și urmărind un procedeu care să asigure atât calitatea înregistrarilor cât și corectitudinea etichetelor atribuite probelor.

Etichetarea bazei de date este în sine un process destul de dificil, deoarece necesită expertiza medicilor specializați în domeniu, neputându-se realiza de către personal necalificat. Procedeul este realizat în mai multe etape, cuprinzând o etapă de interpretare la fața locului, după colectarea probei, urmată de o verificare a posteriori în vederea atestării corectitudinii etichetei, pentru a preveni introducerea de date invalide în baza de date.

Construind pe fundamentele trasate de baza de date Dunstan, baza de date SPLANN extinde acoperirea cu înca două clase de plânset, reprezentative pentru patologie și durerea exercitată de corpul nou-născutului. Baza de date SPLANN conține așadar probe audio atribuite unui număr de șapte clase corespunzătoare nevoilor după cum urmează: colici, eructație, disconfort, foame, durere, durere patologică si oboseală.

Numărul de subiecți care au generat probe crește semificativ, benefiicind de înregistrări ale bebelușilor care ilustrează un singur tip de nevoie. Baza de date SPLANN dispune de un număr de 13373 de plânsete distribuite după cum urmează:

- plânsete reprezentative pentru colici: 225 de probe generate de un număr de 7 bebeluși;

- plânsete reprezentative pentru eructație: 505 de probe generate de un număr de 11 bebeluși;

- plânsete reprezentative pentru disconfort: 2210 de probe generate de un număr de 77 bebeluși;

- plânsete reprezentative pentru foame: 5536 de probe generate de un număr de 92 bebeluși;

- plânsete reprezentative pentru durere: 4404 de probe generate de un număr de 104 bebeluși;

- plânsete reprezentative pentru patologie: 459 de probe generate de un număr de 173 bebeluși;

- plânsete reprezentative pentru oboseală: 34 de probe generate de un singur bebeluș;

Probele au fost colectate și etichetate de către medicii neonatologi implicați în acest proiect, iar mai apoi segmentate și verificate de echipa SOFTWIN împreuna cu membrii ai Laboratorului de Cercetare Speed din cadrul Univerității Politehnica București. Realizarea acestei baze de date, creată în condiții reale, constituie un pas important în dezvoltarea unui sistem automat de recunoaștre și clasificare a plânstelor nou-născuților.

2.3 LUCRĂRI PE ACEEAȘI TEMĂ

Sarcina de clasificare de plânsete de bebeluș a mai fost abordată și înainte în lucrări de cercetare. S-au încercat de asemenea implementări comerciale care însă nu au avut încă aplicații în lumea reală. În cele ce urmează se va prezenta pe scurt fiecare lucrare în parte, detaliindu-se modul în care s-au realizat experimentele împreună cu rezultatele acestora. Un aspect important este faptul că lucrările de cercetare care sunt expuse în continuare folosesc aceeași bază de date care este folosită și în acest proiect și astfel rezultatele sunt comparabile.

- În lucrarea intitulată "Baby cry recognition in real-world conditions", 2016 [4] se încearcă clasificarea plânsetelor de nou născuți folosind baza de date SPLANN, implementând două metode care s-au dovedit eficiente în rezolvarea sarcinilor de recunoaștere de vorbitor și de limbă. Cele două metode sunt Modelul de Amestec Gaussian – Model de Fundal Universal (GMM-UBM) și o tehnică folosind i-vectors. Pentru extragerea de trăsături au fost folosiți 13 coeficienți mel cepstrali (MFCC). Au fost rulate o serie de experimente, folosind pentru clasificare cele șase clase de plânset care sunt folosite și în acest proiect și anume colici, disconfort, eructație, foame, durere și oboseală. Pentru experimentele cu GMM-UBM s-a obținut o acuratețe maxima pe lotul de test de 35%, iar pentru cele care implementează metoda cu i-vectors s-a obținut o acuratețe maximă de 39%. Este de menționat faptul că s-au realizat și experimente folosind o selecție de 3 clase care sunt foarte similare între ele și anume disconfort, foame și oboseală, pentru care s-a obținut o acuratețe de 50.6% pentru GMM-UBM și 58% pentru metoda cu i-vectors.

- Lucrarea "Automated Baby Cry Classification on a Hospital-acquired Baby Cry Database", 2019 [5], abordează sarcina de clasificare a plânsetelor de bebeluș folosind de asemenea un extractor de trăsături, tehnici de selecție a celor mai bune trăsături și nu în ultimul rând o serie de algoritmi de învățare automată din cadrul bibliotecii de învățare automată Weka. Extractorul de trăsături openSMILE deservește drept punct de plecare, folosindu-se un fișier de configurare ce implementează un set de trăsături acustice numit ComParE. Urmează o etapă de selecție a celor mai bune trăsături pentru a înlătura din zgomotul setului mare de trăsături (6373) folosind 3 metode diferite. Prima metodă (Corr), calculează coeficienții de corelație Pearson între fiecare trăsătură din set și clasă. A doua metodă (InfoGain) calculează entropia pentru fiecare trăsătură pentru variabila de ieșire. Ultima metodă de selecție a celor mai bune trăsătsături folosește un algoritm numit Correlation-based Feature Selector (CFS). În urma selecției de trăsături importante se realizază experimente cu 35 de clasificatori, raportându-se rezultatele a doar 18 dintre aceștia, imparțiti în 6 clase, după metodele implementate de fiecare clasificator. S-a determinat că o reducere a numărului de trăsături cu un ordin de mărime poate eficientiza puternic procesul de antrenare, fără a fi în detrimentul performației clasificatorilor. Cel

mai bun rezultat obținut a fost de 51% acuratețe pe lotul de antrenare, pentru un clasificator numit RandomSubSpace.

2.4 INDICI DE PERFORMANȚĂ

Atunci când se dorește evaluarea unui sistem bazat pe rețele neurale este nevoie de o modalitate de măsurare a performanțelor pe care acesta le obține în practică. De ce le mai multe ori, indicii de performanță iau forma unor valori scalare. În cazul în care este nevoie de o modalitate de evaluare a unui sistem de recunoaștere și clasificare de plânset de bebeluș există o serie de măsuri care pot fi folosite. Metricile care sunt folosite pentru a evalua astfel de sisteme trebuiesc calculate folosind un set de date pre-etichetat, pentru a putea diferenția între ceea ce se generează la ieșirea sistemului și ce ar trebui să fie prezis de fapt. În continuare se vor prezenta câțiva indici de performanță utilizati în sarcina de clasificare.

Acuratețea, precizia, reamintirea și scorul F1 sunt măsuri de performanță ce provin din sfera clasificării binare, unde exista 4 tipuri de predicții:

- Adevărat pozitive (AP), atunci când se face o predicție pozitivă pentru un exemplu cu valoare pozitivă.

- Fals pozitive (FP), atunci când se face o predicție pozitivă pentru un exemplu cu valoare negativă.

- Adevărat negative (AN), atunci când se face o predicție negativă pentru un exemplu cu valoare negativă.

- Fals negativ (FN), atunci când se face o predicție negativă pentru un exemplu cu valoare pozitivă.

- Acuratețea este cea mai intuitivă măsură a performanței și este pur și simplu un raport între numărul de predicții corecte și observațiile totale.

$$Acurateţe = \frac{AP + AN}{AP + FP + FN + AN} \tag{2.1}$$

- Precizia este raportul dintre predicțiile adevărat pozitive și totalul predicțiilor pozitive.

$$Precizie = \frac{AP}{AP + FP} \tag{2.2}$$

- Reaminitrea este raportul dintre predicțiile adevărat pozitive și suma predicțiilor adevarat pozitive și fals negative.

$$Reamintire = \frac{AP}{AP + FN} \tag{2.3}$$

- Scorul F1 este o medie ponderată pentru precizie și reamintire, fiind o măsură mai utilă mai ales în cazurile când se lucrează cu seturi de date nebalansate.

$$Reamintire = 2 \cdot \frac{Reaminitire \cdot Precizie}{Reaminitire + Precizie} \tag{2.4}$$

CAPITOLUL 3

NOȚIUNI TEORETICE DESPRE METODE

3.1 SETUL DE TRĂSĂTURI ACUSTICE COMPARE

Reprezintă o colecție de descriptori de nivel scăzut, implementată în fișierele de configurare folosite în cadrul competițiilor Interspeech Computational Paralinguistics ChallengE (ComParE) [6]. Acest set deservește ca punct comun de pornire în rezolvarea diverselor sarcini de clasificare în domeniul recunoașterii vorbirii și al emoției. [7]

Figura 3.1 ilustrează categoriile de descriptori de nivel scăzut prezente în setul ComParE. Un fișier de configurare care implementează acest set a fost folosit în procesul de extragere de trăsături în cadrul acestui proiect.

4 energy related LLD	**Group**
Sum of auditory spectrum (loudness)	prosodic
Sum of RASTA-filtered auditory spectrum	prosodic
RMS Energy, Zero-Crossing Rate	prosodic
55 spectral LLD	**Group**
RASTA-filt. aud. spect. bds. 1–26 (0–8 kHz)	spectral
MFCC 1–14	cepstral
Spectral energy 250–650 Hz, 1 k–4 kHz	spectral
Spectral Roll-Off Pt. 0.25, 0.5, 0.75, 0.9	spectral
Spectral Flux, Centroid, Entropy, Slope	spectral
Psychoacoustic Sharpness, Harmonicity	spectral
Spectral Variance, Skewness, Kurtosis	spectral
6 voicing related LLD	**Group**
F_0 (SHS & Viterbi smoothing)	prosodic
Prob. of voicing	voice qual.
log. HNR, Jitter (local & δ), Shimmer (local)	voice qual.

Figură 3.1 - Categorii de descriptori din setul ComParE [8]

3.2 REȚELE NEURALE ARTIFICIALE

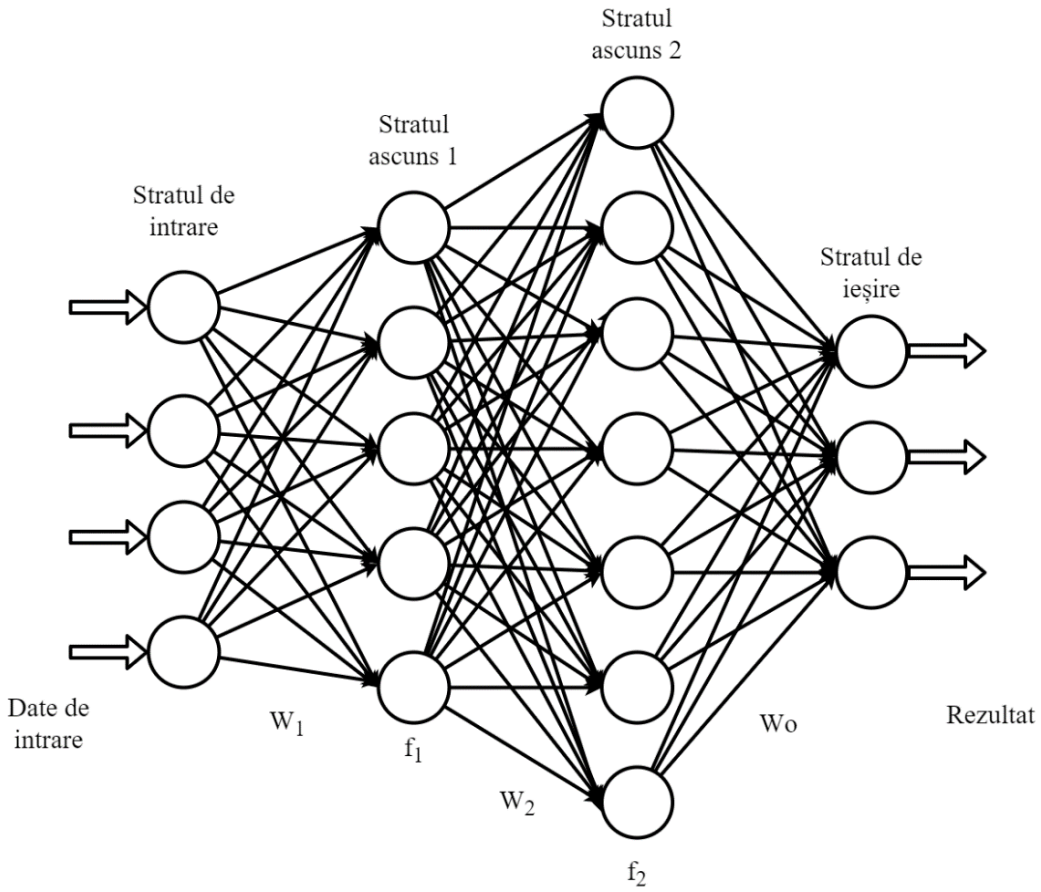
3.2.1 Generalități

O rețea neurală poate fi vazută ca un circuit în cadrul căruia fiecare celulă reprezintă un neuron iar legăturile dintre aceste celule constituie ponderile. Astfel, la nivelul fiecărui neuron se calculează o sumă ponderată a intrărilor, care sunt la rândul lor ieșiri ale neuronilor anteriori. La aceasta suma se adaugă uneori un termen de polarizare. Rezultatul acestei operații de sumare ponderată este apoi trecut printr-o funcție de activare și astfel rezultă ieșirea neuronului curent. La nivel de neuron se poate modela o regresie liniară, la care se introduce o neliniaritate, prin intermediul funcției de activare. [9]

Formula matematică care calculează ieșirea uneui neuron se regăsește în ecuația 3.1, unde *X* este vectorul de intrare, *W* este un vector de ponderi corespunzător neuronului, iar *B* este termenul de polarizare. Acestei ecuații i se aplică o funcție de activare care introduce neliniaritate, fiind în cazul curent o funție numită ReLU (Rectified Linear Unit). În funcție de necesitățile sarcinii, se pot folosi o suită de funcții de activare, care voi fi prezentate în secțiunea *3.2.4*.

$$Z = relu(W^T \cdot X + B) \tag{3.1}$$

Rețeaua neurală primeste la intrare exemple pentru antrenare care pot fi, în funcție de sarcina de învățare, vectori de numere reale, imagini, probe audio etc. Pentru rețelele care sunt bazate pe învațare supervizată, trebuie să fie oferite și rezutatele corecte care ar trebui obținute, adică etichetele exemplelor de antrenare.



Figură 3.2 - Exemplu de rețea neurală artificială

Structura rețelor diferă mult de la caz la caz din punct de vedere architectural, însă se poate face o generalizare în ceea ce privește structura organizată pe straturi. Fiecare rețea are un strat de intrare, responsabil doar cu preluarea datelor, fără să efectueze nicio operație asupra acestora. Acest strat este urmat de unul sau mai multe straturi ascunse, care pot fi de foarte multe tipuri, având funcțiuni diverse în prelucrarea informației și care sunt responsabile cu modelarea procesului de învățare. La final, există un strat de ieșire, responsabil cu preluarea datelor finale și producerea de rezultate într-un format dorit. În figura 3.2 este prezentat un exemplu de rețea neurală artificială, care are un strat de intrare, două straturi ascunse și un strat de ieșire. Stratul de intrare este format din 4 neuroni ce primesc date. Urmează primul strat ascuns, care este format din 5 neuroni, care preiau informatia și o prelucrează introducând valorile de ponderi din W_1, aplicând apoi funcția f_1. Stratul al doilea ascuns efectuează la rândul său prelucrări folosind ponderile din W_2, urmate de aplicarea funcția f_2. La final, stratul de ieșire preia datele provenite în urma ultimul strat ascuns și folosind ponderile din W_o generează rezultatele finale. [10]

Parcursul unui exemplu prin rețea urmat de obținerea unei clasificări se numește propagare în față, iar procedeul invers, prin care se obține rafinarea ponderilor se numește propagare în spate. Pentru învățarea supervizată, o funcție de cost, aleasă conform problemei care se dorește a fi rezolvată, este aplicată rezultatelor obținute, în contrast cu cele corecte prentru a se obține o penalizare a rețelei în cazul unei clasificări eronate. Urmează implementarea unui algoritm de optimizare a cărui funcție obiectiv devine chiar fucția de cost, dorindu-se minimizarea valorii acesteia. Variabilele de stare ale acestui algoritm de optimizare vor fi ponderile și parametrii de polarizare ai neuronilor [9]. Un exemplu de algoritm de optimizare, care este de altfel și unul dintre cele mai folosite, este gradientul descendent.

O iterație este echivalentă cu o parcurgere completă a rețelei pentru un exemplu, atât a etapei de propagare înainte cât și a etapei de propagare înapoi. O epocă definește procesul de trecere prin rețea a tuturor exemplelor de antrenare. În cele mai multe cazuri, procedeul de antrenare necesită mai multe epoci, pentru a se obține minimizarea funcției de cost. Numarul de epoci pentru antrenare este în sine un parametru foarte important, deoarece un numar prea mare de epoci poate determina fenomenul de supraantrenare, care la rândul său însemnă o abilitate redusă de generalizare a rețelei. Astfel, rețeaua va performa foarte bine pe lotul de antrenare, însă va avea rezultate foarte slabe pentru exemple noi.

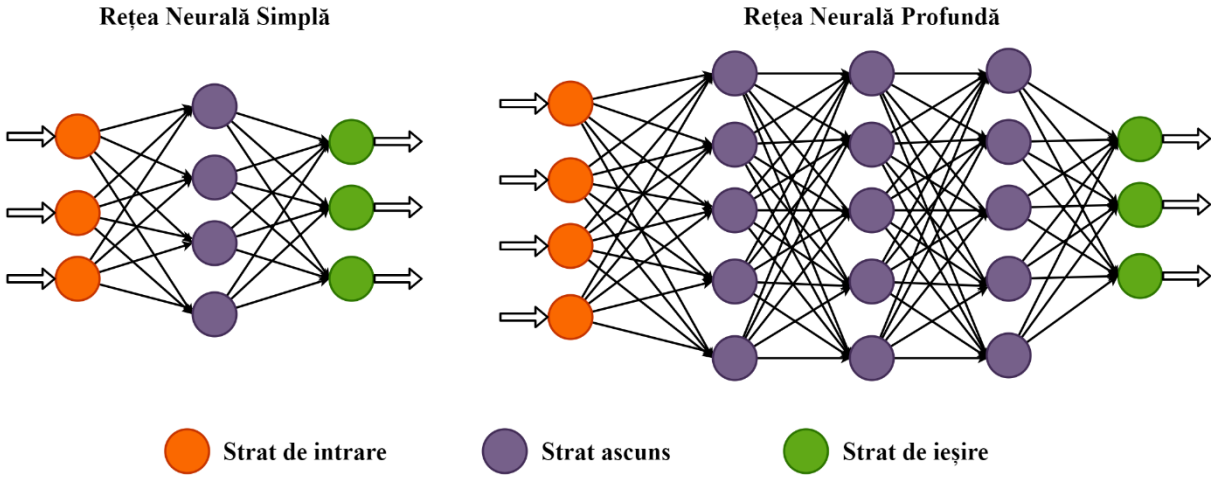
Rețele neurale au un beneficiu semnificativ față de modelele liniare deja existente în domeniul învățării automate. Acestea sunt capabile să modeleze date care nu pot fi separate liniar, folosindu-se de straturile ascunse care introduc elemente de neliniaritate și schimbă modul de reprezentare a datelor, obținându-se o mai bună generalizare a funcției în proces.

Există o suită de tipuri de rețele neurale artificiale, care diferă între ele după tipul de arhitectură, felul în care sunt organizate datele, tipul de straturi ascunse conținute și funcțiunea acestora sau modul de parcurgere a informației prin rețea.

3.2.2 Rețele neurale profunde

Gradul de complexitate ridicat al unor sarcini de recunoaștere și clasificare a determinat dezvoltarea conceptelor de învățare automată către rețele mai mari și mai complicate. Astfel, se introduce noțiunea de rețea neurală profundă, care are mai multe straturi ascunse. Există multe arhitecturi de învățare profundă cum ar fi rețele recurente sau convoluționale, care au fost dezvoltate pentru a rezolva sarcini în domenii precum vedere computerizată, vedere automată, recunoaștere de vorbire, procesare de limbaj natural, recunoaștere audio, filtrare de rețele sociale, traducere automată, bioinformatică și multe altele.

În figura 3.3 este prezentată diferența principală dintre rețele neurale simple și rețele neurale profunde.

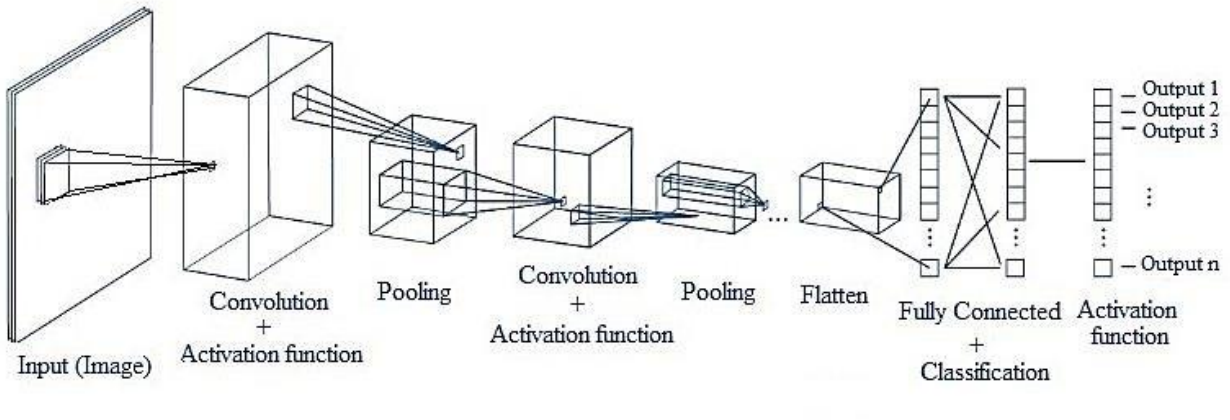


Figură 3.3 - Diferență între tipuri de rețele neurale

3.2.3 Rețele neurale convoluționale

Sunt o clasă de rețele specializate în principal pe analiza imaginilor dar pot avea numeroase aplicații și în alte domenii, fiind de altfel versatile. Similar cu rețelele generale prezentate până acum, acestea sunt formate din straturi cu neuroni care au ponderi și termeni de polarizare doar că în acest caz, fiecare neuron realizează operația de convoluție, la care se adaugă o formă de neliniaritate.

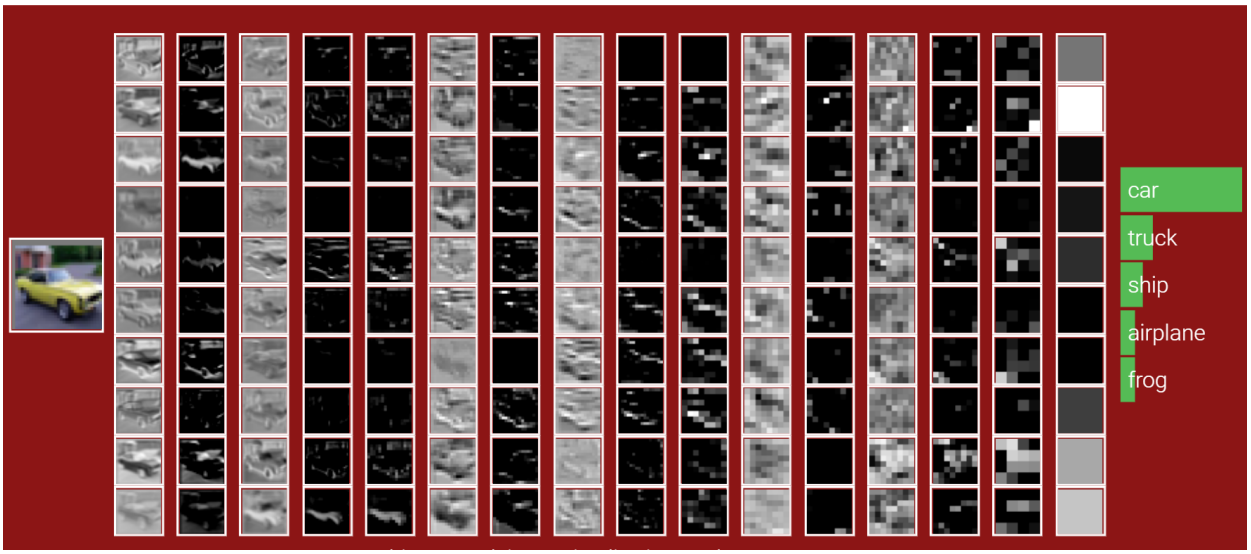
Figura 3.4 prezintă structura de bază a unei rețele neurale convoluționale.



Figură 3.4 - Structura de bază a unei rețele convoluționale [11]

Acest tip de rețele sunt o versiune regularizată a perceptronului multistrat. De obicei, perceptronii multistrat sunt construiți folosind straturi complet conectate, în sensul că fiecare neuron dintr-un strat are legaturi directe cu toți neuronii din stratul următor. Aceasta organizare arhitecturală determină o predispoziție la fenomenul de supraantrenare, o adaptare excesivă pe datele din lotul de antrenare. Pentru a combate această predispoziție, se introuc modalități de regularizare cum ar fi de exemplu adăugarea unei anumite forme de măsurare a magnitudinii ponderilor funcției de pierdere. [12]

Atunci când datele de intrare sunt prea mari, cum se întâmplă adeseori când se lucrează cu imagini, automat numărul de neuroni de intrare va fi de asemenea mare. Rețelele convoluționale implementează o măsură diferită pentru regularizare, fiind avantajate de reprezentarea ierarhică a tiparelor din date. Astfel se indetifică tipare mai mici și mai simple și cu ajutorul acestora se asamblează tipare mai complexe.

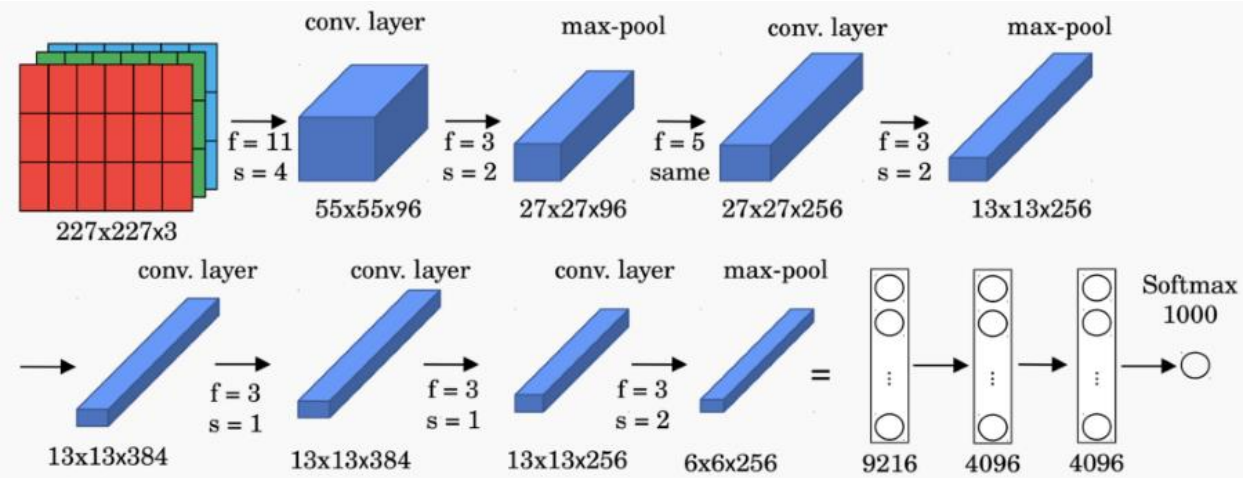


Figură 3.5 - Extragere de trăsături folosind o rețea convoluțională [13]

Figura 3.5 ilustrează descompunerea unei imagini complexe în blocuri constituente mai mici, formate din tipare mai simple. Se poate observa trecerea datelor prin grupări succesive de operații de tipul convoluție, funcție de activare și sondare. La final, trăsăturile se trec printr-o rețea de tipul perceptornului multistrat cu legătrurile complet conectate, în scopul clasificării.

Aceste tipare se pot extrage folosind o serie de operații realizate de mai multe ori. Operațiile sunt de cele mai multe ori covoluții, aplicări de funcții de activare și în final sondări. Spre deosebire de convoluțiile folosite în analiza de imagni, unde ponderile nucleelor sunt fixe și au un scop predefinit, în cazul învățării automate, convoluțiile au nuclee cu ponderi ce pot fi modificate pe parcursul antrenării. Astfel, se poate realiza o învățare a caracteristicilor constituente ale datelor de intrare, în funcție de relevanța acestora pentru rezolvarea sarcinii de clasificare. Pentru reducerea dimensionalității datelor se folosesc straturi de sondare, care selecteaza în funcție de anumite criterii predefinite, informațiile cele mai relevante. [14]

Figura 3.6 exemplifică reducerea diemnsionalității pornind de la o imagine color de dimensiune 227x227 pixeli.



Figură 3.6 - Reducerea dimensionalității într-o rețea convoluțională [15]

Rețelele convoluționale pot fi folosite și în cazul în care se dorește o clasificare a probleor audio, de exemplu în scopul clasificării de plânset de bebeluș. O astfel de sarcină poate fi abordată folosind ca exemple pentru antrenarea rețelei spectrograme extrase din fișierele audio.

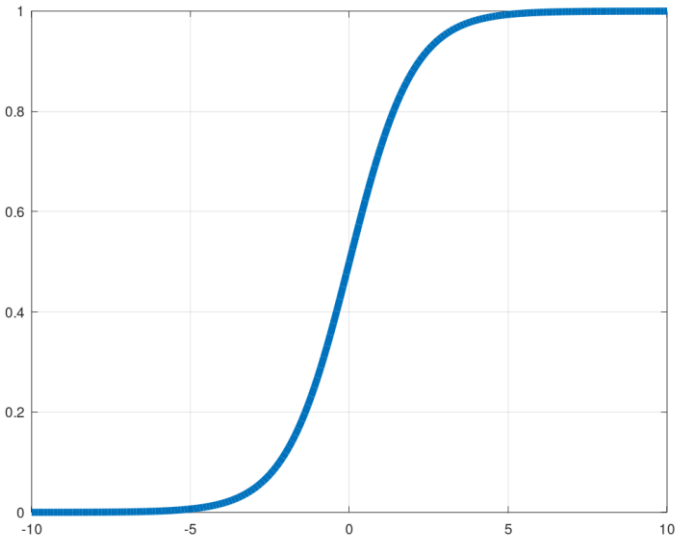
3.2.4 Funcția de activare

Pentru a putea modela date neseparabile liniar, se aplică în procesul de antrenare așa numitele funcții de activare care introduc proprietăți de neliniaritate. În funcție de cerințele specifice ale sarcinii care se dorește a fi rezolvată și de performanțele obținute, se pot alege o varitetate de funcții de activare. În continuare se vor prezenta cele mai populare opțiuni de funcții de activare.

3.2.4.1 Funcția Sigmoid

Funcția sigmoid ia valori între 0 și 1, având carateristică o curbă în froma de "S". Caracterul neliniar îi determină utilitatea în situații când aproximarea este dificilă. Pe măsură ce datele de intrare de îndepărtează de valoarea centrală 0, ieșirea funcției tinde la 1 sau 0, panta fiind aproape dreaptă. Acest fapt are ca și consecințe fenomenul de dispariție a gradienților, care este una dintre problemele semnificative ale rețelelor. În ceea ce privește consumul de resurse, această funcție este destul de costisitoare.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

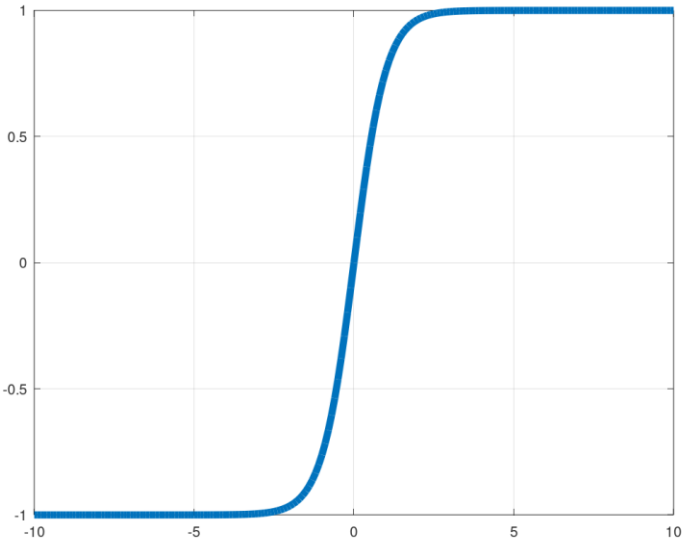


Figură 3.7 - Graficul funcției sigmoid

3.2.4.2 Funcția tangentă hiperbolică

Această funcție are un caracter neliniar, fiind de ajutor în cazul aproximărilor complexe, fiind similară funcției sigmoid, dar luând valori între -1 și 1. Funcția tangentă hiperbolică suferă de aceeași problemă a dispariției gradienților, însă este mai avantajoasă dotorită centrării în 0 pe axa oy, fiind capabilă de modelarea datelor cu o puternică polarizare pozitivă, neutră sau chiar negativă. Computațional vorbind, este de asemenea destul de costisitoare.

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.3}$$

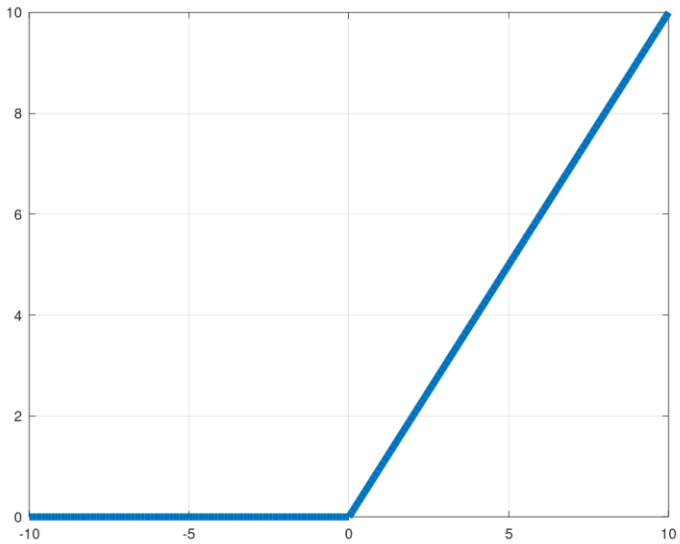


Figură 3.8 - Graficul funcției tangentă hiperbolică

3.2.4.3 Funcția ReLU

Se numește și funcție de rectificare liniară. Aceasta nu poate lua valori negative însă poate lua valori pozitive foarte mari, nefiind limitată. Funcția ReLU este de asemena neliniară. Această funcție se află printre cele mai populare alegeri de funcții de activare deoarece este imună la problema dispariției gradienților și este eficientă în ceea ce privește consumul de resurse.

$$relu(x) = \max(0, x) \tag{3.4}$$

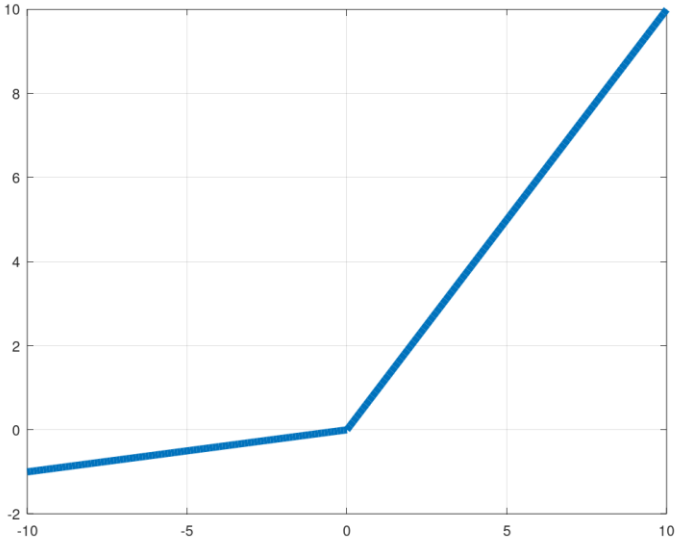


Figură 3.9 - Graficul funcției ReLU

3.2.4.4 Funcția Leaky ReLU

Leaky ReLU este o variantă de funcție de rectificare liniară care permite un gradient mic de valori pozitive, atunci când unitatea nu este activă. Avantajele folosirii acestui tip de funcție de activare constau în evitarea problemei blocării învățării atunci când gradientul este 0, cum este posibil să se întâmple în cazul folosirii funcției ReLU.

$$leaky\ relu\ (x) = \max(0.1 \cdot x, x) \tag{3.5}$$

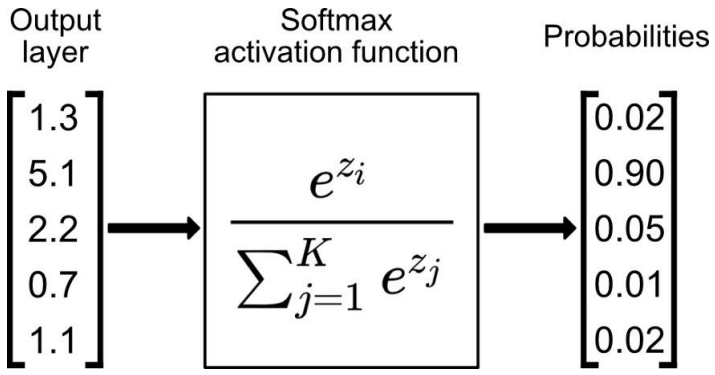


Figură 3.10 - Graficul funcției Leaky ReLU

3.2.4.5 Funcția softmax

Această funcție este folosită la sfârșitul lanțului de procesare, oferind probabilitățile de apartenență la clasele posibile, cu valori subunitare.

$$softmax\,(z_i) = \frac{e^{z_i}}{\sum_j^K e^{z_j}} \tag{3.6}$$



Figură 3.11 - Transformarea valorilor folosind Softmax [16]

3.2.5 Funcția de cost

Funcția de cost este responsabilă cu penalizarea unei rețele atunci când aceasta obține rezultate eronate. Învățarea realizată de o rețea constă în rafinarea unor ponderi și găsirea parametrilor potriviți pentru sarcina de rezolvat, în sensul generării unei predicții bune. Astfel, funcția de cost devine practic funcția obiectiv pentru un algoritm de optimizare, în sensul minimizării erorii.

Având în minte aceste considerente, alegerea unei funcții de cost potrivită pentu sarcina rețelei devine un aspect esențial. Deoarece funcția cost comprimă un sistem relativ complex la o singură valoare scalară, alegerea unei funcții potrivite este uneori dificilă, având în vedere nevoia captării trăsăturilor cele mai importante pentru modelare.

Există mai multe tipuri de funcții de cost, care sunt relevante în cazul fiecărei sarcini de predicție sau clasificare. Câteva dintre cele mai importante astfel de funcții sunt expuse în cele ce urmează.

- Eroarea pătratică medie (Mean Square Error): este relevantă în cazul în care este nevoie de o ieșire constituită dintr-o singură valoare scalară, cum este de altfel și cazul regresiilor liniare. Formula erorii pătratice medii se regăseste în ecuația 3.7.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \widehat{Y_l}\right)^2 \tag{3.7}$$

- Entropia încrucișată (Cross Enthropy Loss): se poate folosi atunci când sarcina de rezolvat a rețelei constă în clasificare binară sau pe mai multe clase. Ecuația 3.8 dă formula de calcul.

$$J(w) = -\frac{1}{N}\sum_{n=1}^{N}[y_n \log \widehat{y_n} + (1 - y_n)\log(1 - \widehat{y_n})] \tag{3.8}$$

3.2.6 Algoritmul de optimizare

În ceea ce privește algoritmul de optimizare folosit în minimizarea funcției de cost, există câteva tipuri principale care pot fi folosite, diferențiate de metoda de lucru după cum urmează:

- Algoritmi derivativi de ordinul întâi, care se folosesc în metoda de optimizare de gradientul funcției. Cea mai populară alegere în cadrul acestei categorii este metoda gradientului negativ.
- Algoritmi derivativi de ordinul al doilea, în cadrul cărora se folosește matricea Hessiană în procesul de optimizare. Matricea Hessiană conține derivatele parțiale de ordin doi ale câmpului funcției. Dacă se cunoaște derivata a doua, calculul folosind această metodă este mai rapid, în caz contrar, apar costuri mari în ceea ce privește resursele folosite.
- Algoritmi nederivativi, în cadrul cărora nu se folosește deloc informație provenită din gradientul funcției.

3.2.6.1 Metoda gradientului negativ

Metoda gradientului negativ este bazată pe un algoritm de optimizare multidimensional derivativ de oridinul întâi, care actualizează valorile proderilor folosind gradientul funcției de cost, valorile actuale și un pas constant.

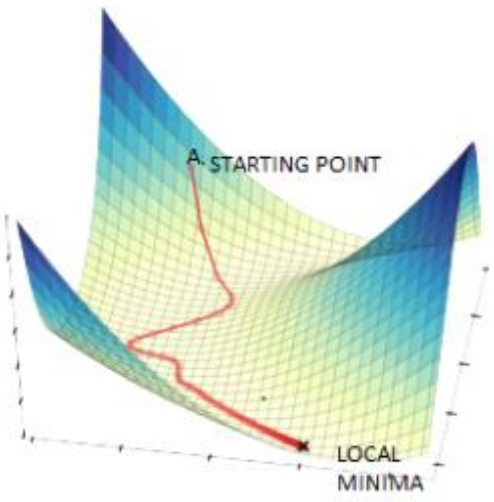
$$w_{k+1} = w_k - \alpha \nabla \mathrm{J}(w_k) \tag{3.9}$$

Ecuația 3.9 prezintă formula de calcul pentru metoda gradientului negativ. Vectorul de ponderi *w* este actualizat de la valoarea anterioară folosind gradientul funcției de cost J, care este înmulțit cu un parametru α, numit și rată de învățare, care constituie pasul de actualizare. În funcție de magnitudinea parametrului α, care este o constantă, există posibilitatea de a se sări peste valoarea de optim, în cazul selecției unei valori prea mari. În cazul opus, când valoarea aleasă este prea mică, timpul de căutare al punctului de optim poate crește semnificativ, fapt nedorit în contextul unei antrenări rapide.

Metoda gradientului negativ se poate aplica fie folosind câte un exemplu odată, fie folosind o împărțire a setului de antrenare pe loturi. În cazul al doilea, actualizarea ponderilor se realizează folosind mai multe exemple în același timp.

Aplicarea metodei gradientului negativ pe loturi este mai eficientă deoarece aceasta atinge convergența într-o manieră mai rapidă și mai stabilă față de cazul când se folosește un singur exemplu. Folosirea unui singur exemplu determină un timp de calcul crescut, dată fiind instabilitatea actualizării funcției de cost, în ciuda convergenței la puncte de minim. Versiunea de algoritm care folosește un singur exemplu se numește gradient descendent stochastic (SGD).

Figura 3.12 ilustrează procedeul de căutare a unui minim local folosind metoda gradientului descendent într-un context multidiemnsional.



Figură 3.12 - Optimizare cu gradient descendent [17]

Performanțele metodei gradientului descendent pot fi îmbunătățite folosind o serie de mecanisme concepute să crească stabilitatea convergenței funcției de cost.

3.2.6.2 Impulsul

Una dintre metoldele de îmbunătățire a performanțelor algoritmului SGD este impulsul. Acesta presupune introducerea în ecuație a unui temen menit să ia în calcul o parte din valorile obținute la pasul anterior de optimizare. Astfel, se poate obține o aliniere mai bună cu direcția potrivită de descreștere, stabilizându-se valorile funcției de cost atunci când este necesară schimbarea direcției. Din acest proces rezultă o convergență mai previzivbilă. [18]

Formula 3.10 oferă modalitatea de calcul a vectorului de actualizare, având vectorul calculat la pasul anterior. Se poate observa prezența termenului γ, numit și termen de impuls, care exprimă măsura aportului pe care îl are vectorul calculat la pasul anterior în etapa actuală. De regulă se aleg valori subunitare pentru γ, cum ar fi 0.9.

$$V(t) = \gamma V(t-1) + \alpha \nabla J(w_k) \tag{3.10}$$

Vectorul rezultat se introduce în formula 3.11 și astfel se obține regula de actualizare a ponderilor.

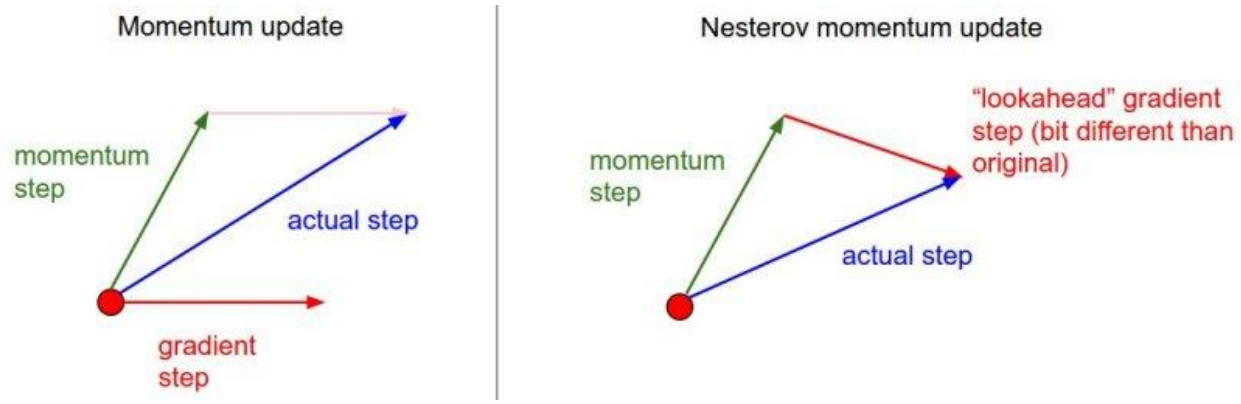
$$w_{k+1} = w_k - V(t) \tag{3.11}$$

3.2.6.3 Gradientul accelerat Nesterov

Este o versiune ușor modificată a îmbunătățirii aduse de introducerea impuluslui în ecuație care a început să prindă recent popularitate. Acesta vine ca o soluție pentru o problemă ce apare în cazul utilizării impuluslui simplu. În anumite cazuri, există posibilitatea apariției unei acumulări de impuls care poate duce la ratarea minimului, pe măsura evoluției funcției, dacă punctul de optim se află într-o regiune foarte abruptă. Gradientul Accelerat Nesterov este capabil să încetinească parcursul astfel încât să nu se rateze punctul de minim, adaugând totodată un grad sporit de adaptablitate la schimbări ale funcției.

$$V(t) = \gamma V(t-1) + \alpha \nabla J\big(w_k - \gamma V(t-1)\big) \tag{3.12}$$

Modul de funcționare al GAN implică o aproximare a poziției viitoare în care se va găsi algoritmul precum și a ponderilor pe care le va avea. Formula de actualizare este redată în ecuația 3.12, iar funcționarea este exemplificată în figura 3.13.



Figură 3.13 - Actualizarea GAN [19]

3.2.6.4 Adam

Adam (Adaptive Moment Estimation) este un algoritm de optimizare care are ca și caracteristică principală divizarea ratei de învățare pentru a putea urmări mai bine evoluția fiecărui parametru în parte. În contextul implementărilor care folosesc această metodă, rata de învățare este actualizată într-o manieră adaptivă, relativ la gradienții care au fost calculați din valorile anterioare ale ponderilor. În formulele următoare m_t și v_t reprezintă momentele statistice ale gradienților, mai exact media și varianța necentrată.

$$\widehat{m_t} = \frac{m_t}{1 - \beta_1} \tag{3.13}$$

$$\widehat{v_t} = \frac{v_t}{1 - \beta_2} \tag{3.14}$$

$$w_{k+1} = w_k - \frac{\alpha}{\sqrt{\widehat{v_t} + \varepsilon}} \widehat{m_t} \tag{3.15}$$

Fiind una dintre cele mai populare opțiuni în materie de algoritmi adaptivi, Adam performează admirabil, reușind să conveargă destul de repede. Alte versiuni de algoritmi adaptivi întâmpină probleme cum ar fi fluctuația valorilor funcției de cost sau dispariția gradineților însă atunci când se folosește Adam, aceste probleme sunt evitate. [20]

3.2.7 Metode de regularizare

Unul dintre principalele aspecte ce trebuiesc urmărite cu atenție în procesul de antrenare al unei rețele neurale este evitarea fenomenului de supraantrenare. Deși rețelele neurale excelează în rezolvarea sarcinilor ce implică modelarea de date neseparabile liniar, acestea sunt predispuse să supraîncadreze și astfel sa nu poată generaliza pentru exemple care nu au fost prezente în lotul de antrenare. Acest fenomen poate fi ameliorat prin implementarea metodelor de regularizare.

3.2.7.1 Regularizarea funcției de cost

Presupune introducerea unui temen de regularizare în interiorul funciției de cost, pentru a trunchia valorile prea mari ale ponderilor. Termenul de regularizare este folosit împreună cu matricea de ponderi sub diverse forme.

În continuare vor fi expuse principalele variante de regularizare a funcției de cost.

- Varianta cu normă de tip L1, care foloseste matricea de ponderi în modul:

$$J_{actual}(W) = J(W) + \lambda \sum_{i=1}^{n} |w_i| \tag{3.16}$$

- Varianta cu normă de tip L2, care folosește pătratul matricei de ponderi:

$$J_{actual}(W) = J(W) + \lambda \sum_{i=1}^{n} w_i^2 \tag{3.17}$$

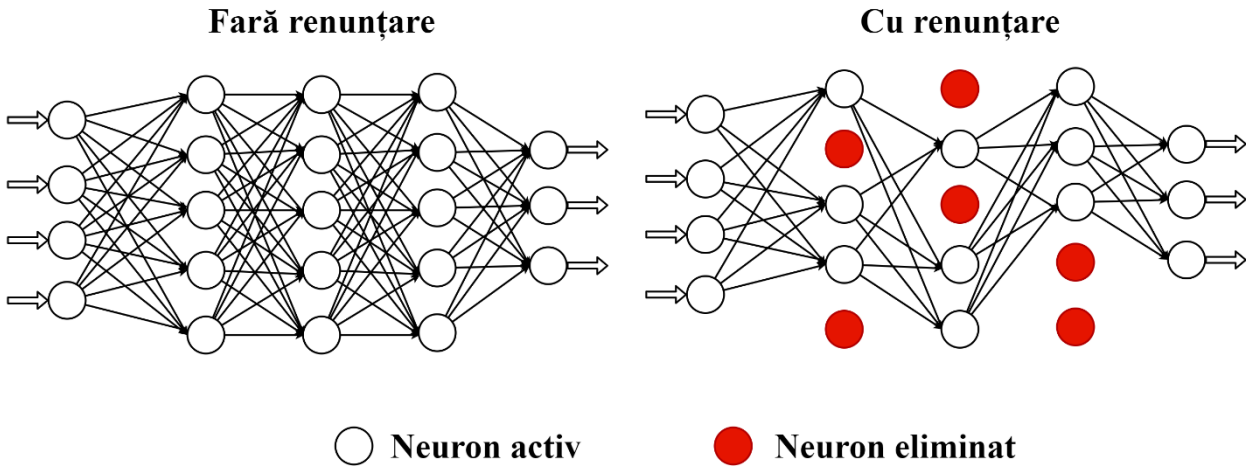
- Varianta care folosește o combinație a celor două prezentate anterior, alături de un coeficient de regularizare adițional:

$$J_{actual}(W) = J(W) + \lambda_1 \sum_{i=1}^{n} |w_i| + \lambda_2 \sum_{i=1}^{n} w_i^2 \tag{3.18}$$

3.2.7.2 Regularizarea celulelor neurale

Printre mecanismele de regularizare se numără și o tehnică extrem de populară ce implică renunțarea la o parte din celulele neurale cunoscută și sub numele de dropout. Implementarea acestei metode de regularizare determină o independența crescută a celulelor neurale, datorită lipsei de intrări de la celulele anterioare. Există un parametru care reprezintă procentul de legături alese în manieră aleatorie la care se renunța în timpul antrenării numit rată de renunțare. Această rată constituie în sine un hiperparametru și alegerea unei valori potrivite poate avea consecințe favorabile în procesul de antrenare, mai ales în condițiile evitării fenomenului de supraantrenare.

Figura 3.14 ilustreaza aplicarea tehnicii de renunțare.



Figură 3.14 - Tehnica de renunțare

3.2.8 Considerente de natură problematică

Procesul de proiectare a unei rețele neurale profunde implică o serie de aspecte problematice, datorită complexității crescute în raport cu tehnicile de învățare automată clasice. Rețele neurale profunde sunt mai puternice în ceea de privește abilitatea de generalizare pentru sarcini mai complexe, însă acest avantaj vine la pachet cu o suită de dificultăți. [21]

În primul rând, cu cât dimensiunea rețelei crește, cu atât va fi nevoie de mai multă putere de procesare și mai multe resurse de memorie, dat fiind volumul foarte mare de parametri care

trebuiesc învățați și ponderi care trebuiesc rafinate. În anumite cazuri se poate ajunge la numere imense de prametri, de ordinul zecilor de milioane, fapt care determină timpi de antrenare foarte mari, uneori de câteva zile. Pentru a combate această deficiență, s-au dezvoltat paltforme de paralelizare care au permis antrenarea modelelor direct pe plăcile grafice, scazând considerabil timpul de antrenare.

Proiectrea unei rețele neurale profunde capabile să rezolve eficient o sarcină complexă depinde de foarte multe aspecte. Pentru început, trebuie aleasă o arhitectura potrivită, ceea ce implică alegerea tipurilor de straturi precum si a numărului acestora. Urmează alegerea funcțiilor de activare, a funcției de cost și a algoritmului de optimizare. Toți acești pași sunt foarte importanți deoarece introduc parametri cu impact profund asupra rezultatelor obținute de modele.

Aceste variabile se numesc hiperparametri și au o natură problematică, deoarece procesul de reglaj este de multe ori unul de încercare și eroare. Cele mai întâlnite variabile cu statut de hiperparametru sunt numărul de straturi prezente în rețea, dimensiunea acestora, rata de învățare sau număul de epoci pentru antrenare. S-au dezvoltat mecanisme de reglaj automat și folosirea acestora introduce un strat adițional de abstractizare într-un context deja foarte complex.

Rețelele neurale profunde au nevoie de foarte multe date etichetate pentu a putea generaliza cu succes sarcini complexe, evitând fenomenul de supraantrenare. De multe ori, obținerea unei baze de date suficient de voluminoasă pentru a putea antrena eficient o rețea este un proces dificil. Și pentru această problemă s-au găsit soluții ce includ de exemplu manipulări ale datelor deja existente pentru a genera noi exemple, cum se poate spune despre translatări, rotiri sau oglindiri în cazul imaginilor. [22]

CAPITOLUL 4

TEHNOLOGII UTILIZATE

4.1 LIMBAJUL DE PROGRAMARE PYTHON

Acesta a jucat un rol foarte important in dezvoltarea proiectului, fiind de altfel limbajul de programare în care au fost scrise sistemele de clasificare. Folosirea acestuia este în principal motivată atât de accesibilitatea și viteza cu care se pot realiza aplicații care folosesc concepte avansate cât și de disponibilitatea bibliotecilor gratuite dedicate învațării automate. Implementarea rețelelor neurale este mult mai facilă folosind biblioteci precum: TensorFlow, Keras, Pytorch sau Scikit-learn.

Python este un limbaj de scriptare, ceea ce îi conferă avantajul de fi ușor de folosit și înteles, dată fiind natura sintaxei profund abstractizată. Fiind prietenos cu începătorii, după doar câteva ore de învățare, oricine poate programa în Python. Codul este scris dinamic, ceea ce impune folosirea unei indentări standardizate, fapt care contribuie la îmbunatățirea lizibilității. Ne putem da seama ce ar trebui să facă un program doar citind codul acestuia. [23]

De asemenea, natura de limbaj interpretat este un beneficiu. Codul de Python nu trebuie compilat, acesta fiind intern convertit într-o formă intermediară numită bytecode. Acest aspect elimina dificultăți cum ar fi legarea corectă la biblioteci. Depanarea problemelor din cod este mai facilă în Python, codul fiind executat linie cu line, astfel localizarea problemei fiind mai rapidă.

Python are caracterisitca cheie de a fi un limbaj multi-paradigmă, ceea ce însemnă că se pot combina mai multe abordări cum ar fi programarea orientată pe obiecte sau programarea

procedurală. Acest fapt consituie un avantaj în construirea unui sistem complex, unde se îmbina module de naturi diferite, cum ar fi un sistem de recunoaștere și clasificare care folosește rețele neurale.

Popularitatea de care acesta dă dovadă în rândul dezvoltatorilor de software este de asemenea un beneficiu semnificativ, acesta bucurându-se de o comunitate numeroasa, gata să contribuie în rezolvarea problemelor și crearea de noi biblioteci și implementări în numeroase domenii. Fiind foarte flexibil, Python poate fi folosit în multe contexte de dezoltare. Acesta intefațează foarte bine cu alte limbaje de programare, ceea ce îi extinde acoperirea de aplicabilitate în domenii precum dezvoltarea de jocuri video, exploatarea datelor și inteligența artificială sau chiar aplicații web.

În ceea ce privește lucrul cu bilbioteci externe, gestionarea se realizează foarte facil utilizând aplicații precum Pip sau Conda. Aplicațiile de gestionare de pachete sunt responsabile atât cu descărcarea și instalarea bibliotecilor cât și cu menținerea dependențelor necesare și versionarea corectă. Este de menționat că Python ofera o colecție generoasă și extensivă de funcții în biblioteca standard.

Acest limbaj de programare a fost folosit în cadru proiectului la scrierea modulelor ce consituie sistemele de clasificare. Astfel, folosind Python au fost scrise modulele responsabile cu încărcarea și preprocesarea datelor cât și cele responsabile cu definirea, antrenarea și eveluarea modelelor. Multe dintre aceste module folosesc funcționalități provenite din bibliotecile de algoritmi de rețele neurale.

În continuare vor fi prezentate bibliotecile de învățare automată care au fost folosite în elaborarea proiectului.

4.2 BIBLIOTECA PYTORCH

Este o bibliotecă de învățare automata gratuită și open source, utilizată de regulă în domenii precum computer vision și natural language processing. Aceasta a fost dezvoltată în principal de către laboratorul de cercetare în inteligentă artificială al companiei Facebook. A fost lansată pentru prima dată în 2016, aflându-se în mentenanță și dezvoltare pâna în prezent. O suită de aplicații software ce folosesc tehnici de Deep Learning au la bază biblioteca Pytorch, fapt care atestă relevanța acesteia în contextul actual al domeniului. Printre aceste aplicații se numără și Autopilotul companiei de automobile electrice Tesla sau limbajul de programare probabilistă Pyro al companiei Uber. [24]

Pytorch beneficiază de două caracteristici importante și anume:

- procesare de tensori (similară cu cea regasită in cadrul bibliotecii NumPy), cu accelerare puternică prin intermediul unităților de procesare grafică (GPU);
- rețele neurale profunde bazate pe un sistem de diferențiere automata;

Unul dintre modulele importante prezente în această bibliotecă este Autograd. Astfel, sunt înregistrate operațiile care au fost realizate, ca mai apoi acestea să fie parcurse în sens invers pentru a calcula gradienții. Construirea rețelelor neurale beneficiază de folosirea acestei metode deoarece se economisește timp într-o epocă, prin calcularea diferențierii parametrilor la pasul de propagare în față prin rețea.

Un alt modul interesant care se regăsește în această bibliotecă este modulul Optim. Acesta este responsabil cu implementarea numeroșilor algoritmi de optimizare, utilizați în construirea rețelelor neurale. Multe dintre metodele clasice de optimizare sunt deja conținute de această biblioecă, dezvoltatorul fiind astfel scutit de nevoia de a le scrie de la zero.

De asemenea, modulul nn merită menționat, acesta ajutând în procesul de dezvoltare a modelului, făcând definirea grafurilor computaționale mult mai facilă.

În contextual dezvoltării proiectului prezent, biblioteca Pytorch a fost folosită în principal la construirea rețelelor neurale profunde cu arhitectura personalizată (în cadrul experimentelor cu MLP), incluzând definirea, antrenarea, evaluarea, salvarea și încărcarea modelelor. Este de menționat faptul că pentru experimentele cu arhitectura TabNet s-a folosit o implementare în Pytorch.

4.3 BIBLIOTECA SCIKIT-LEARN

Scikit-learn este de asemenea o biblioteca gratuită și open source care asigură utilizatorilor o suită de de algoritmi de învățare automata atât supervizată cât și nesupervizată prin intermediul unei interfețe coerente în Python. Dezvoltarea acesteia a început ca un proiect al lui David Cournapeau, ca o extesie pentru pachetul SciPy și a fost lansat pentru prima oară în 2007. [25]

Ulterior, codul sursă a fost rescris de către o echipa de dezvoltatori de la Institutul francez de cercetare în Automatică și Calculatoare, fiind mai apoi lansat public pentru prima data în 2010. Biblioteca Scikit-learn este foarte populară și bine menținută, aflându-se în dezvoltare până în prezent, de către un grup de 30 de contribuitori (la data redactării acestei lucrări).

Această bibilotecă este focalizată în principal pe modelarea de date [26]. Câteva dintre cele mai populare grupuri de modele pe care le oferă sunt:

- Clustering: pentru grupare de exemple fără etichetă, de exemplu KMeans;
- Validare încrucișată: pentru estimarea performanței modelelor nesupervizate pe seturi de date noi;
- Seturi de date: pentru seturi de date de test și pentru generarea de seturi de date cu proprietăți specific pentru investigarea comportamentului modelului;
- Reducerea dimensionalității: pentru reducerea numărului de atribute din date pentru sumarizare, vizualizare și selecție de trăsături cum ar fi PCA (Principal Component Analysis – Analiza componentelor principale);
- Metode de asamblare: pentru combinarea predicțiilor mai multor modele supervizate;
- Extragere de trăsături: pentru definirea atributelor din imagini și text;
- Selecție de trăsături: pentru identificarea atributelor semnificative din care se pot crea modele supervizate.
- Reglarea parametrilor: pentru a folosi cât mai eficient modelele supervizate;
- Învătare multiplă: pentru sumarizarea și descrierea datelor multi-dimensionale complexe;
- Modele supervizare: cuprinde o gamă largă, care nu se limitează la modele liniare generalizate, analiză discriminantă, naïve bayes, metode lazy, rețele neurale, mașini cu support vectorial și arbori de decizie;

În cadrul proiectului actual, această bibliotecă a fost folosită extensiv, deservind în mai multe etape ale expereimentelor cu LinearSVC cum ar fi preprocesarea datelor și împarțirea seturilor de date, definirea, antrenarea, optimizarea și evaluarea modelului, precum și în partea de reprezentare a rezultatelor (matrice de confuzie). De asemenea, numeroase componente ale acestei biblioteci au fost utilizate și în cadrul experimentelor cu TabNet.

În contiuare, se va expune platforma Google Colab, prin intermediul căreia au fost dezvoltate și rulate experimentele prezentate în această lucrare.

4.4 PLATFORMA GOOGLE COLABORATORY

Platforma Google Colaboratory (sau Colab, pe scurt) este un mediu de dezvoltare în cloud gratuit, conceput de Google Research, pentru a facilita construirea de proiecte axate pe învățare automata, analiză de date și educație. Astfel, oricine poate scire și executa cod de Python prin intermediul unui browser. În spatele acestei platforme se află defapt un serviciu de Jupyter notebook, hostat pe calculatoarele Google, care nu necesită instalare sau configurare, oferind acces gratuit la resurse de calcul care includ și plăci de procesare grafică.

Diferența dintre această platformă și Juyter este că cea din urmă este un proiect open source pe care Colab se bazează. Astfel se oferă posibilitatea de a folosi și împărtăși notebook-uri cu alți dezvoltatori, fără a fi nevoie de descărca, instala sau rula alte programe, în afară de un simplu browser de internet.

Notebook-urile Colab pot fi stocate prin intermediul serviciului Google Drive, folosind un cont de utilizator, sau pot fi încărcate direct de pe platforma GitHub. Acestea pot fi descărcate în format ipynb sau împărtășite cu ușurința, existând de asemenea posibilitatea de încarcare de notebook-uri Jupyter preexistente local.

Codul este executat în cadrul unei mașini virtuale private, legate de contul utilizatorului. Mașinile virtuale sunt înlătruate atunci cand nu manifestă activitate timp de o anumită perioadă de timp, și au o durata de viață maximă limitată de către serviciul Colab la 12 ore, pentru a conserva și eficientiza folosirea de resurse. Ficărei sesiuni i se aloca initial o cantitate de memorie, care se poate extinde ulterior, după nevoile sepcifice ale fiecărui notebook. Utilizatorul are de asemenea posibilitatea de a alege tipul de sesiune, în funcție de acceleratorul hardware dorit, care poate fi de tip CPU sau GPU. Gestiunea resurselor se face automat și transparent fața de utilizator, fapt ce determină o ușurință sporită de folosire.

Fiind o platforma de dezvoltare susținută de una dintre cele mai mari companii de tehnologie, aceasta beneficează de o integrare foarte bună cu multe biblioteci pentru învățare profundă, cum ar fi PyTorch, TensorFlow, Keras sau OpenCV.

Dezvoltarea programelor se face foarte simplu și eficient, utilizând celule de două tipuri: de text și de cod. Natura modulară a notebook-urilor permite o impmentare rapidă a blocurilor constituente ale sistemelor de clasificare. Folosind celule de text se pot delimita și documenta diversele module prezente, contribuind la o structură clară și concisă a proiectelor.

Datorită acestor considerente de natură tehincă și practică, platforma Google Colab a fost aleasă pentru dezvoltarea și rularea experimentelor prezentate în această acestei lucrări.

În continuare, va fi expus extractorul de trăsături openSMILE, acesta fiind relevant în generarea unei baze de date în format CSV.

4.5 EXTRACTORUL DE TRĂSĂTURI OPENSMILE

Pentru generarea bazei de date pe care s-au realizat experimentele a fost folosit un extractor de trăsaturi audio open source intitulat openSMILE (open-Source Media Interpretation by Large feature-space Extraction), fiind dezvoltat de o companie din Munich. Acest proiect există încă din 2008, aflându-se în mentenanța companiei germane audEERING GmbH încă din 2013.

Extractorul este în principal destinat recunoașterii de emoție și este foarte folosit în procesarea afectivă, în sfera comunității de cercetători. Acesta este folosit atât în zona de cercetare academică, cât și în aplicații comerciale pentru a analiza semnale de vorbire și muzică în timp real. Spre deosebire de recunoașterea automată de vorbire care extrage conținutul comunicat dintr-un semnal

de vorbire, openSMILE are abilitatea de a recunoaște caracteristicile unui semnal audio muzical sau de vorbire. Câteva astfel de caracteristici care sunt codificate în vorbirea umană sunt de exemplu emoțiile, vârsta, sexul și personalitatea precum si stările în care se află vorbitorul, cum ar fi depresia, starea de ebrietate sau chiar tulburările patologice vocale. Programul este capabil să clasifice de asemenea caracteristici ale semnalului muzical cum ar fi de exemplu starea de spirit emanată de o melodie și recunoașterea de elemente constituente cum ar fi segmente de refren, game, acorduri, tempo sau genul muzical.

Extractorul openSMILE este scris in C++ pur, având o arhitectură fexibilă, rapidă și eficientă, putând fi rulat pe majoritatea platformelor cum ar fi Linux, Windows și MacOS. Acesta a fost conceput pentru procesare online în timp real, dar poate fi folosit și local în modul pe loturi pentru a procesa seturi mari de date, cum a fost și cazul prezentei lucrări.

Pentru a facilita interoperabilitatea, openSMILE este capabil să citească și să scrie diverse formate de date folosite în mod obișnuit în domenii precum exploatarea datelor și învățare automată. Aceste formate includ:

- PCM WAVE (Pulse-code modulation Waveform Audio File Format) pentru fișiere audio;
- CSV (Comma Separated Values, format spreadsheet);
- ARFF (Attribute-Relation File Format, exploatare de date cu biblioteca Weka) pentru fișiere de date text;
- HTK (Hidden-Markov Toolkit), fișiere de parametri;
- Un format de matrice binară de tip float, pentru date cu caracteristici binare;
- Stream-uri video prin intermediul bibliotecii openCV;

O trăsătură interesantă a extractorului este că datorită arhitecturii puternic modularizată, aproape toate datele intermediare generate în timpul procesului de extragere (cum ar fi ferestrele de date audio, spectrele, etc.) pot fi accesate și salvate în fișiere pentru vizualizare sau procesări ulterioare.

Date fiind facilități oferite, extractorul openSMILE a fost folosit pentru generarea bazei de date în format CSV, pornind de la fișierele WAV cu plânsete de bebeluși din baza de date SPLANN.

Se va expune în continuare platforma de calcul paralel CUDA, relevantă în procesul de încărcare de date și antrenare a modelelor, în cadrul expreimentelor prezentate.

4.6 PLATFORMA CUDA

CUDA (Compute Unified Device Arhitecture) este o platformă pentru calcul paralel ce include atât hardware cât și software sub forma unui API (Application Programming Interface), dezvoltată de către compania americană de procesoare grafice Nvidia. Folosind plăcile grafice pentru sarcini ce necesită o cantitate masivă de calcul, se poate obține o accelerare semnificativă în ceea ce privește timpul de execuție. Astfel, platforma CUDA este un strat software care oferă acces direct la setul virtual de instrucțiuni și la elmentele pentru calcul paralel ale unității de procesare grafică, pentru execuția nucleelor de calcul. [27]

Platforma CUDA a fost dezvoltată astfel încat să lucreze cu limbaje de programare precum C, C++ sau Fortran. Această marcă de accesibilitate facilitează folosirea de resurse GPU pentru specialiștii în programare paralelă, spre deosebire de API-urile dezvoltate anterior cum ar fi Direct3D sau OpenGL, care necesitau abilități avansate în progamare grafică.

În ultimii ani, dezvoltarea tehnologică a permis avansuri semnificative în domenii precum învățarea automată, data science, analiza numerică sau imagistica medicală, datorită implementărilor cu calcul paralel, pe procesoare grafice Nvidia, utilizând platforma CUDA.

Bibliotecile de învățare profundă moderne (cum este și cazul celor folosite în cadrul acestei lucrări) beneficiază de implememtări CUDA, care permit antrenarea modelelor de rețele neurale profunde direct pe unitățile de procesare grafică. În experimentele prezentate în contextul proiectului actual a fost folosită impementarea CUDA din Pytorch, pentru a încărca datele direct pe plăcile graficie disponibile prin intermediul platformei Google Colab, obținându-se astfel viteze de antrenare de până la 40 de ori mai mari decât în cazul în care antrenarea modelelor se făcea pe CPU.

CAPITOLUL 5

METODE PROPUSE

5.1 ARHITECTURI CU EXTRACTOR DE TRĂSĂTURI

Această lucrare propune realizarea mai multor experimente bazate o structură similară, puternic modularizată, pentru a putea permite o dezvoltare rapidă și simplă.

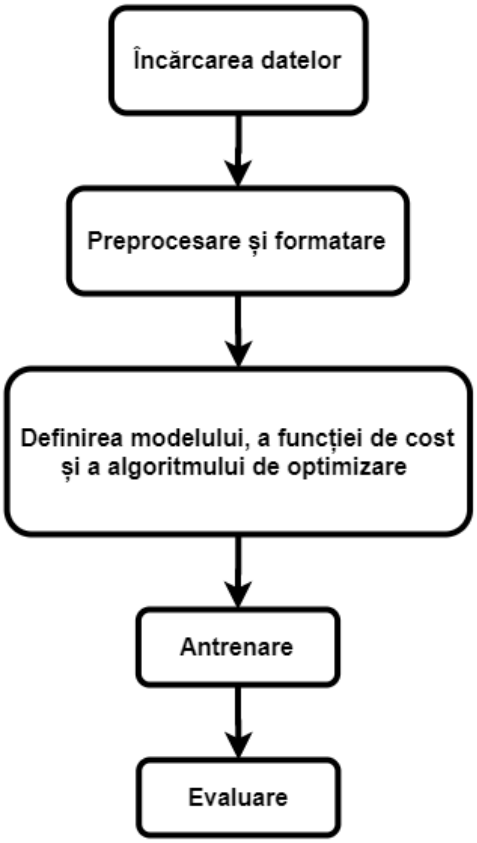
Pentru început, se va folosi un extractor de trăsături pentru a genera o bază de date în format CSV. În lanțul de procesare, prima etapă este reprezentată de încărcarea datelor, de care va fi responsabil un modul care va fi folosit în toate experimentele. Urmează blocuri responsabile cu preprocesarea și formatarea datelor, în funcție de fiecare experiment în parte, fiind nevoie de anumite reglaje. În această etapă se va realiza de asemenea împărțirea bazei de date în seturi de antrenare și de test.

Următoarele blocuri din lanțul de procesare sunt responsabile cu definirea arhitecturii rețelei neurale folosite, împreună cu definirea funcțiilor de cost, respectiv a algoritmului de optimizare folosit. În acest fel, se vor defini trei modele cu moduri de funcționare diferite. Primul experiment va implementa o arhitectură personalizată, pornind de la modelul perceptronului multistrat (MLP), al doilea experiment va implemneta un clasificator linair de tip mașină cu suport vectorial (LinarSVC), iar ultimul experiment va folosi un clasificator bazat pe învățare tabulară interpretabilă atentă (TabNet). După ce aceste blocuri și-au atins scopul se poate trece la etapa de antrenare, care este de asemenea diferită pentru fiecare experiment.

Apoi există blocuri ce permit salvarea și încărcarea de modele antrenate în funcție de intenția utilizatorului. Urmează etapa de testare și evaluare a modelelor antrenate, care se realizează ușor

diferit pentru fiecare experiment. La final există o secțiune cu ajutorul căreia se pot afișa rezultatele obținute de modele sub diverse forme cum ar fi tablele, grafice sau matrici de confuzie.

În principal această structură 5.1 este respectată pentru toate cele trei experimente realizate cu mențiunea că în anumite cazuri unele blocuri mai puțin importante iau o altă formă sau nu sunt prezente.



Figură 5.1 - Structura generală a sistemelor dezvoltate

5.2 MLP

Primul experiment propune implementarea unei arhitecturi de rețea neurală profundă bazată pe modelul perceptronului multistrat în scopul realizării sarcinii de clasificare.

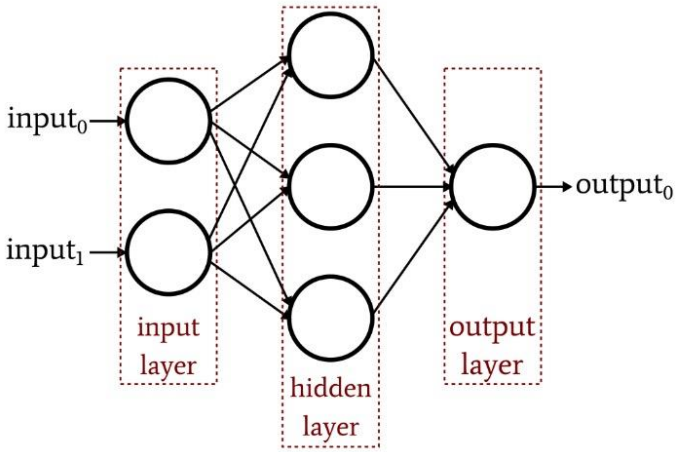
Perceptronul multistrat reprezintă una dintre cele mai simple forme de rețea neurală care se poate realiza având de regulă straturi complet conectate. Modelul de bază are un strat de intrare, un singur strat ascuns și în final un strat de ieșire. Pornind de la această arhitectură se va dezvolta o rețea cu mai multe straturi.

Modelul propus are un strat de intrare de dimensiunea datelor pe care se face antrenarea. Acest prim strat nu efectuează nicio operație asupra vectorilor de intrare, fiind responsabil doar cu preluarea informației pentru a putea fi prelucrată de straturile următoare.

Urmează apoi un număr de straturi ascunse complet conectate care au dimensiuni reglabile. Fiecare neuron dintr-un strat ascuns efectuează operația de sumare ponderată la care se adaugă un termen de polarizare. Rezultatul acestei procesări este trecut printr-o funcție de activare ce introduce aspecte de ne-liniaritate, contribuind la modelarea datelor neseparabile liniar. Se va experimenta cu diverse configurații arhitecturale, atât în ceea ce privește numărul de straturi cât și dimensiunea lor.

Odată ce datele au fost trecute prin straturile ascunse, ele vor ajunge la stratul de ieșire, care are dimensiune egală cu numărul de clase în care se face clasificarea.

Figura 5.2 expune arhitectura generică a perceptronului multistrat.



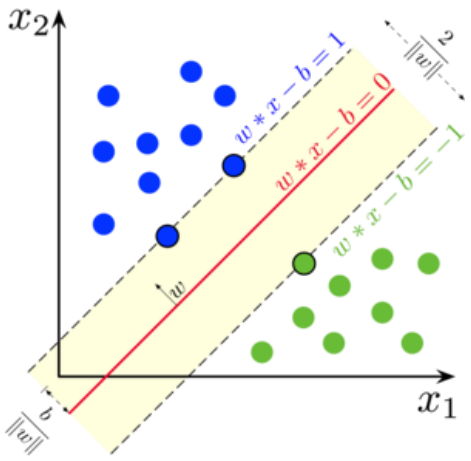
Figură 5.2 - Perceptron multistrat [28]

5.3 LINEARSVC

Cel de al doilea experiment presupune implementarea unei arhitecturi de rețea neurală bazată pe o mașină cu suport vectorial. Deși în general mașinile cu suport vectorial sunt folosite pentru a performa clasificare liniară, ele sunt capabile de a realiza clasificare neliniară destul de eficient, folosind un nucleu special. Astfel se poate realiza o modelare a datelor cu o dimensionalitate ridicată, cum este și cazul acestei lucrări.

Din punct de vedere formal, o mașină cu suport vectorial construiește un hiperplan sau un set de hiperplane într-un spațiu cu multe dimensiuni, care poate fi folosit pentru sarcini de clasificare sau predicție. Principiul de funcționare al clasificatorilor SVM este bazat pe găsirea unor puncte numite vectori de suport, care constituie parametri pentru formarea suprafeței de separare [29].

Figura 5.3 exemplifică principiul enunțat mai sus.



Figură 5.3 - Separare folosind vectori de suport [30]

Deși acest experiment nu se bazează pe tehnici de învățare profundă, el este relevant deoarece poate oferi informații utile cu privire la felul în care se pot modela exemplele de din baza de date, în formatul actual, cu trăsături extrase.

Acest clasificator va fi folosit pentru a evidenția măsura în clasele din baza de date în format CSV cu trăsături extrase pot fi separate liniar două câte două. Mai multe detalii despre acest proces vor fi oferite în secțiunea *7.2.2*.

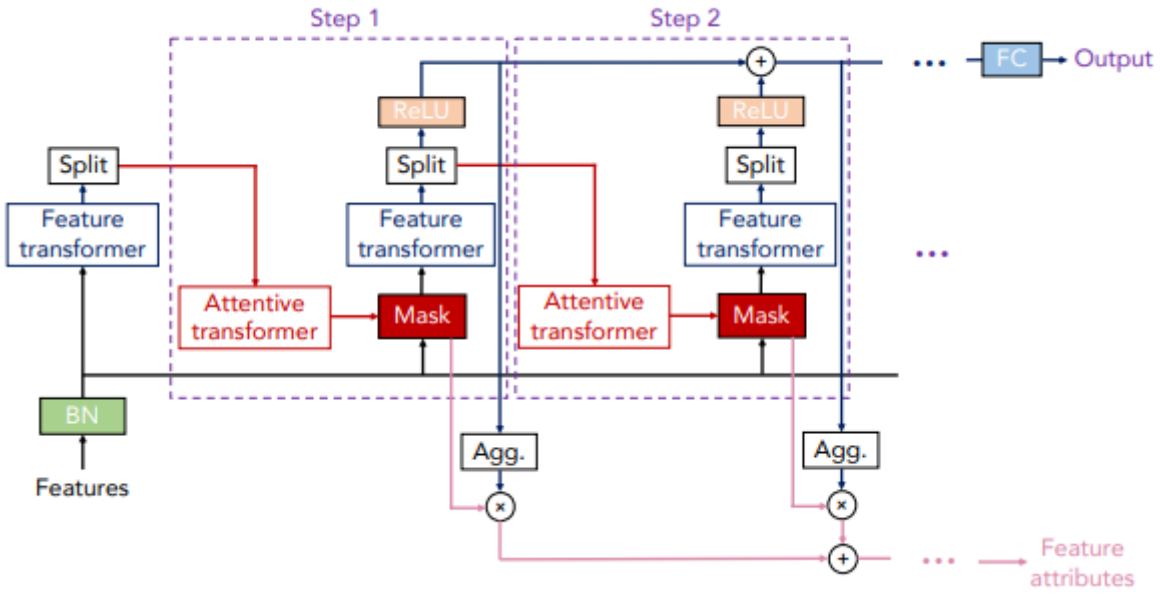
5.4 TABNET

Structura actuală a bazei de date generate cu ajutorul extractului de trăsături poate fi privită ca un exemplu de date tabulare, deoarece fiecare exemplu are același număr de trăsături, adică fiecărei linii din tabel îi corespund același număr de coloane. Pornind de la această premiză ultimul experiment își propune să exploreze viabilitatea unui sistem bazat pe o arhitectură ce folosește tehnici de învățare profundă pe seturi de date cu structură tabulară.

Motivele pentru care tehnicile clasice de învățare profundă nu dau rezultate foarte bune în acest context includ faptul că de foarte multe ori trăsăturile construite din date tabulare sunt corelate, ceea ce duce la situația în care un subset de trăsături este responsabil pentru majoritatea puterii de predicție. Un alt motiv este legat de bazele de date care sunt puternic debalansate cum este și cazul baze de date folosită în cadrul acestui proiect. Sarcina devine mult mai dificilă atunci când există o disproporție semnificativă între numărul de exemple din fiecare clasă [31].

Deși în general tehnicile de învățare profundă nu sunt foarte eficiente atunci când se folosesc seturi de date care au format tabular, descoperirile recente din domeniu au permis surclasarea algoritmilor bazați pe de arbori de decizie, care erau cei mai performați.

TabNet este o rețea neurală care a fost concepută special pentru a învăța din seturi de date tabulare. structura acesteia este foarte diferită față de alte rețele care folosesc o versiune modificată de tip hibrid bazată pe arbori de decizie. Caracteristica cheie a rețelei este o mască antrenabilă a trăsăturilor de intrare. Codificatorul prezentat în figura 5.4 presupune o secvența de etape de decizie care codifică și selectează trăsături prin intermediul măștii antrenabile.



Figură 5.4 - Codificatorul rețelei TabNet [32]

CAPITOLUL 6

DETALII DE IMPLEMENTARE

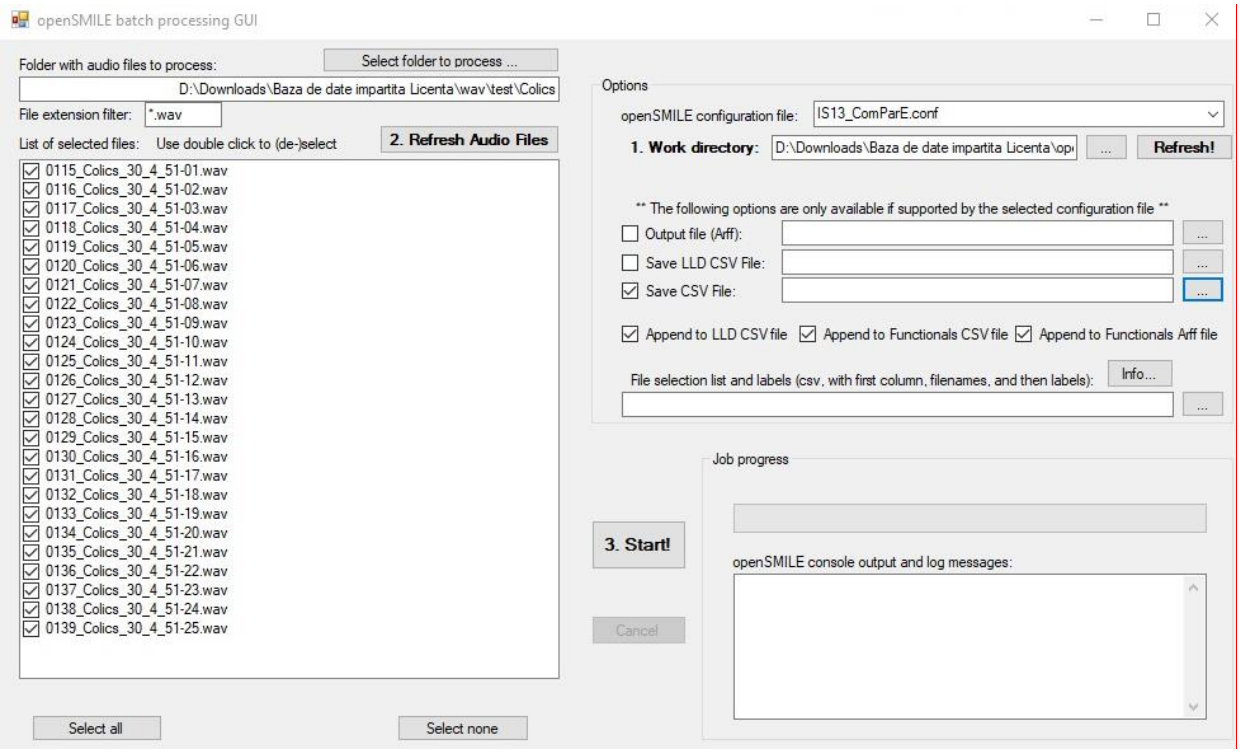
6.1 EXTRAGERE DE TRĂSTURI

Procesul de extragere de trăsături începe cu repartizarea fișierelor de tip WAV, puse la dispoziție de către laboratorul SpeeD, în direcotare diferite după nevoia pe care o reprezintă. De asemenea, este de menționat faptul că separarea pe loturi de antrenare și test a fost realizată folosind aceeași împărțire ca în cazul lucrărilor anterioare care abordează această temă, din considerente de comparație. Pe scurt, această împărțire respectă o proporție de 90% din datele prezente pentru setul de antrenare și 10% pentru setul de test, având în vedere numărul de exemple din fiecare clasă (fiecare clasă are 10% din exemplele aferente atribuite lotului de test).

Odată ce fișierele au fost separate în directoare, s-a folosit extractorul de trăsături openSMILE, prezentat anterior, în versiunea cu interfață grafică pentru a procesa mai multe fișiere WAV într-un lot. Pentru procesul de extragere s-a folosit un fișier de configurare care implementează setul de trăsături acustice ComParE-2013 (6373 trăsături). Au rezultat astfel 14 fișiere în format CSV, fiecare conținând date de antrenare și respectiv de test pentru următoarele nevoi: colici, disconfort, ercutație, foame, durere, oboseală și durere patologică. Ulterior s-a renunțat la folosirea fișierelor reprezentative pentru durere patologică, pentru a avea rezultate comparabile cu lucrările de cercetare anterioare care au lucrat pe aceeași bază de date.

În figura 6.1 este ilustrată interfața grafică a extactorului de trăsături openSMILE iar figura 6.2 deomonstrează cum arată exemple extrase dintr-un fișier CSV, având pe prima linie tipul de

caracteristică extras, iar mai apoi fiind listate trăsăturile extrase din fiecare fișier pe rând, numele fișierului original fiind conținut în prima coloană.



Figură 6.1 - Interfața grafică openSMILE

```
name;frameTime;audspec_lengthL1norm_sma_range;audspec_lengthL1norm_sma_maxPos;auds
'0001_Colics_8_6_1-01.wav';0.000000;1.317413e+000;5.268817e-001;0;5.855656e-001;1.
'0002_Colics_8_6_1-02.wav';0.000000;2.570976e+000;3.076923e-001;0;1.769539e+000;1.
'0003_Colics_8_6_1-03.wav';0.000000;1.473857e+000;5.087720e-001;0;1.079422e+000;1.
'0004_Colics_8_6_1-04.wav';0.000000;2.904706e+000;8.823529e-001;0;1.307903e+000;1.
'0005_Colics_8_6_1-05.wav';0.000000;4.379579e+000;7.538462e-001;1.538462e-002;7.00
'0006_Colics_8_6_1-06.wav';0.000000;2.012293e+000;7.341772e-001;0;1.771937e+000;2.
'0007_Colics_8_6_1-07.wav';0.000000;3.047776e+000;5.882353e-001;0;2.142119e+000;2.
'0008_Colics_8_6_1-08.wav';0.000000;2.319407e+000;4.210526e-001;0;1.110577e+000;1.
'0009_Colics_8_6_1-09.wav';0.000000;2.560218e+000;6.901408e-001;2.816901e-002;8.63
'0010_Colics_8_6_1-10.wav';0.000000;2.568320e+000;8.725490e-001;0;8.992715e-001;1.
'0011_Colics_8_6_1-11.wav';0.000000;1.475347e+000;4.339623e-001;0;1.375066e+000;1.
'0012_Colics_8_6_1-12.wav';0.000000;2.402212e+000;4.453125e-001;1.562500e-002;1.15
'0013_Colics_8_6_1-13.wav';0.000000;9.272983e-001;1.194030e-001;5.671642e-001;1.74
'0014_Colics_8_6_1-14.wav';0.000000;2.044401e+000;1.944444e-001;1.851852e-002;1.45
'0015_Colics_8_6_1-15.wav';0.000000;1.311135e+000;2.207792e-001;3.896104e-002;6.02
'0016_Colics_8_6_1-16.wav';0.000000;2.710335e+000;1.683168e-001;4.009901e-001;4.84
'0017_Colics_8_6_1-17.wav';0.000000;2.315171e+000;6.438356e-001;0;1.929054e+000;2.
'0018_Colics_8_6_2-01.wav';0.000000;3.224159e+000;4.675325e-001;1.298701e-002;1.52
'0019_Colics_8_6_2-02.wav';0.000000;2.242179e+000;3.235294e-001;0;2.307882e+000;2.
'0020_Colics_8_6_2-03.wav';0.000000;2.205568e+000;6.727273e-001;0;1.805326e+000;2.
'0021_Colics_8_6_2-04.wav';0.000000;2.326138e+000;3.437500e-001;0;2.173199e+000;2.
'0022_Colics_8_6_2-05.wav';0.000000;2.520874e+000;4.375000e-001;0;1.771976e+000;2.
```

Figură 6.2 - Exemple în format CSV

Baza de date în format CSV generată a fost mai apoi urcată pe cloud, folosind Google Drive, pentru a putea fi accesată prin platforma Google Colab.

6.2 ÎNCĂRCAREA DATELOR

Începând de la această etapă, toată activitatea se desfășoară pe platforma Google Colab, folosind instrucțiuni Python. Din motive de protecție a datelor, accesul la baza de date aflată pe Google Drive se face în mod securizat, fiind nevoie de un token de confirmare pentru acces. Pentru a putea rula programele de pe o stație nouă, este nevoie de o copie a bazei de date în contul de Google Drive atașat utilizatorului curent. Rularea următoarelor instructiuni va prompta utilizatorul să intre pe o pagină de autenticare, unde îi va fi generat un token care trebuie copiat în casuța aferentă din notebook. Odată ce autentificarea a fost realizată cu succes și utilizatorul a introdus codul token, se poate trece la etapa următoare.

```
from google.colab import drive
drive.mount('/content/drive')
```

Pentru încarcarea datelor din fișierele CSV a fost concepută funcția loadWith() (codul sursă se gaseste în Anexa 1), care are ca parametru de intrare variabila shuffle, care determină dacă încărcarea se face amestecând exemplele sau nu. Pentru funcționarea corectă, este nevoie de setarea unei căi de acces relativă la rădăcina unității de stocare de pe Google Drive.

```
path = '/content/drive/My Drive/Colab Notebooks/extracted_features_csv/'
```

Funcția loadWith() creeaza două array-uri de tip numpy goale care vor deservi drept unități de acumulare pentru datele care vor fi alipite succesiv. Cele două variabile sunt train_data și test_data. În continuare se va lua fiecare fișier CSV pe rând și se va procesa individual, fiind convertit într-un format compatibil numpy. Pentru îndeplinirea acestei sarcini se folosește subrutina processAndAppend() care are parametri de intrare indicele clasei care trebuie procesată, precum si vectorul în care se face încărcarea. Pentru citirea efectivă din fișierele CSV se folosește funcția read_csv() din biblioteca Pandas cu anumiți parametri cum ar fi de exemplu numele fișiererului din care se citește și tipul de delimitator folosit. Citirea se face într-o variabilă de tip buffer numită data. Se realizează conversia în array numpy folosind metoda to_numpy(). Sunt eliminate primele două coloane din fiecare exemplu datorită redundației informatiei conținute în acestea (numele fișierului original și doar valoarea de 0). Se adaugă de asemenea eticheta crespunzătoare fiecărei clase la sfârșitul vectorului. Datele vectorului preexistent și daca este cazul (variabila shuffle este setată) acestea sunt amestecate.

```
def processAndAppend(cls, old_data):
  data = pd.read_csv(filename, delimiter=';', engine='c', dtype='a')
  data = data.to_numpy()
  data = data[:, 2:]
  label = np.zeros((data.shape[0], 1)) + cls
  data = np.append(data, label, axis=1)
  print("\tappended data shape = " + str(data.shape))
  if shuffle:
    np.random.shuffle(data)
  new_data = np.append(old_data, data, axis=0)
  return new_data
```

Funcția processAndAppend() este apelată pentru fiecare clasă succesiv, ocupându-se atât de fișierele de antrenare cât și de cele de test.

```
clases = ['colics', 'discomfort', 'eructation', 'hunger', 'pain', 'tiredn
ess']

  for cls in range(0, 5):
    print('Importing Class ' + str(cls) + '/5 examples...')
    filename = path + 'train_' + clases[cls] + '.csv'
    train_data = processAndAppend(cls=cls, old_data=train_data)
    filename = path + 'test_' + clases[cls] + '.csv'
    test_data = processAndAppend(cls=cls, old_data=test_data)
    print("\ttrain_data shape = " + str(train_data.shape))
    print("\ttest_data shape = " + str(test_data.shape))
```

Odată ce este rulată celula, codul funcției loadWith() este încărcat în memorie și poate fi aplelat de oriunde altundeva din cadrul notebook-ului. Funcția a fost gândită astfel încât să permită încărcarea personalizată, în funcție de nevoile utlizatorului fiind posibilă selecția doar a anumitor clase, în orice combinație.
Modulul care cuprinde acest script a fost folosit în toate experimentele sub aceeași formă.

6.3 PREPROCESARE ȘI FORMATARE

Următoarele celule sunt responsabile cu preprocesarea datelor și formatarea acestora, astfel încât să poată interfața bine cu rețele neurale. Pentru început se vor împărți array-urile cu date de antrenare și de test în vectori separați pentru intrare în rețea și pentru etichete (prepoziția np_ identifică elementul ca fiind un array de tip numpy). Această celula este prezentă în toate experimentele.

```
np_X_train = np_train_data[:, :-1]
np_Y_train = np_train_data[:, -1]
np_X_test = np_test_data[:, :-1]
np_Y_test = np_test_data[:, -1]
```

Urmează un modul responsabil cu diverse transformări ce sunt aplicate succesiv vectorilor listați mai sus, însă doar pentru experimentul cu arhitectura de tip MLP. Se vor prezenta succint tipurile de transformări pentru un singur vector.

Prima instrucțiune transformă tipul de date stocat în array-uri în formatul numpy float64, pentru extinderea preciziei de calcul.

```
np_X_train = np_X_train.astype(np.float64)
```

Pornind de le vectorul de tip numpy cu date reprezentate în formatul float64 se va genera un tensor folosind funcția from_numpy(), regăsită în cadrul bibliotecii PyTorch.

```
X_trainTensor = torch.from_numpy(np_X_train)
```

Pentru vectorii de etichete este necesară conversia de la tensori simpli la tensori de tip Long.

```
Y_trainTensor = Y_trainTensor.type(torch.LongTensor)
```

Se vor asambla apoi tensorii cu exemple precum și cei cu etichete în elemente de tip TensorDataset, folosind de asemenea metode din utilitarul bibliotecii Pytorch. Aceste elemente sunt necesare pentru construirea unor structuri responsabile cu aprovizionarea datelor în rețea,

numite DataLoaders, în cadrul cărora se pot specifica parametri precum dimensiunea loturilor folosite în procesul de antrenare pe loturi.

```
train_data = torch.utils.data.TensorDataset(X_trainTensor, Y_trainTensor)

trainset = torch.utils.data.DataLoader(train_data, batch_size=64, shuffle
=True)
```

Există și o celulă menită să ajute în procesul de gestionare a resurselor de memorie, dar aceasta este opțională, neavând vreun efect în funcționarea sistemului. În cadrul acestei celule se pot marca variabilele redundante folosind instrucțiunea del, ca mai apoi acestea sa fie colectate de către serviciul gunoier din Python.

```
del np_X_train
gc.collect()
```

De menționat este faptul că pentru experimentul cu TabNet s-a realizat atât împărțirea menționată mai sus cât și o împărțire adițională cu set de validare, folosind o funcție din biblioteca Scikit-learn, despre care se va detalia în *Capitolul 7*. Parametrul stratify specifică faptul că împărțirea se va realiza în mod proporțional cu numărul de exemple din ficare clasă, folosind ca valoare vectorul de etichete corespunzător lotului de antrenare.

```
X_train, X_valid, Y_train, Y_valid = train_test_split(X_train, Y_train, t
est_size=0.1, random_state=42, stratify=Y_train)
```

6.4 DEFINIRE MODEL

Următorul bloc din lanțul de procesare este responsabil cu definirea arhitecturii rețelei, precum și a configurării funcțiilor de cost și a algoritmilor de optimizare. Pentru fiecare experiment, acest bloc este diferit și se vor prezenta individual detalii de implementare.

6.4.1 Definire MLP

Definirea modelului se face sub forma unei clase, folosind metode din bibilioteca Pytorch. S-a experimentat cu mai multe arhitecturi, însă acestea vor fi discutate în *Capitolul 7*. În continuare se va prezenta doar un exemplu de implementare.

În prima parte a definiției clasei se specifică structura pe straturi în formă secvențială, modelând parcursul informației prin rețea. Următoarea instrucțiune definește primul strat din rețea care realizează o transformare liniară. Acesta are 6373 de parametri de intrare, de la care pleacă 256 de legături spre următorul strat. Pentru a preveni fenomenul de supraantrenare se recurge la implementarea metodei de regularizare ce presupune renunțarea la o parte din neuroni, în funcție de un parametru pasat funcției din biblioteca Pytorch și anume rata de renunțare. Sunt apoi definite celelalte straturi, în ordine secvențială.

```
  def __init__(self):
    super(Net, self).__init__()

    self.fc1 = nn.Linear(6373, 256)
    self.dropout1 = nn.Dropout(0.3)
    self.fc2 = nn.Linear(256, 256)
```

```
        self.dropout2 = nn.Dropout(0.2)
        self.fc3 = nn.Linear(256, 6)
        self.dropout3 = nn.Dropout(0.2)
```

Cea de-a doua parte a definiției clasei este responsabilă cu descirerea funcției de propagare înainte. Aici se implementează succesiunea de straturi precum și funcțiile de activare care în acest caz sunt de tip ReLU. La final se returnează predicțiile.

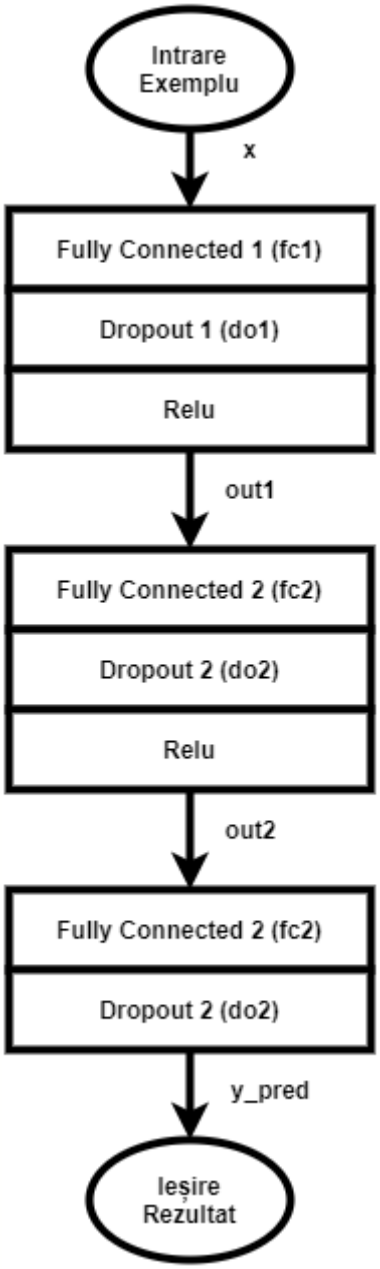
```
def forward(self, x):
    do1 = self.dropout1(self.fc1(x))
    out1 = F.relu(do1)

    do2 = self.dropout2(self.fc2(out1))
    out2 = F.relu(do2)

    do3 = self.dropout3(self.fc3(out2))
    y_pred = do3

    return y_pred.float()
```

Astfel, la finalul definiției modelului exemplificat mai sus, acesta are următoarea structură:



Figură 6.3 - Exemplu structură rețea MLP

Urmează o celulă responsabilă cu definirea funcției de cost și a algoritmului de optimizare. Este de menționat faptul că s-a folosit entropia încrucișată împreună cu un parametru pentru ponderile claselor. Pentru a îmbunătăți abilitatea de generalizare a rețelei, s-a optat pentru o modalitate de echilibrare a setului de date. În acest scop este creat un vector de ponderi care conține valori invers proporționale numărului de exemple al fiecărei clase din totalul de exemple.

```
weights = [1/2.189, 1/17.178, 1/3.906, 1/42.711, 1/34.014, 1/0.257]
class_weights = torch.FloatTensor(weights).cuda()
criterion = nn.CrossEntropyLoss(weight=class_weights, reduction='mean')
```

Pentru algoritmul de antrenareare s-a optat pentru Adam. Se setează de aici ținta și anume parametrii rețelei împreună cu hiperparametrul rată de învățare.

```
optimizer = torch.optim.Adam(net.parameters(), lr=1e-3)
```

Odată ce aceste module sunt rulate, rețeaua este configurată și încarcată în memorie și se poate începe procesul de antrenare.

6.4.2 Definire LinearSVC

Pentru exemperimentul cu LinearSVC, procesul de definire a modelului este mult mai simplu, dată fiind implementarea în biblioteca de învățare automată Scikit-learn. O singură instrucțiune cu grad înalt de abstractizare este suficientă pentru a configura clasificatorul, odată ce fișierele bibliotecii au fost importate în proiect.

```
clf = make_pipeline(StandardScaler(), LinearSVC(random_state=0, C=1, clas
s_weight='balanced', max_iter=1000))
```

Funcția make_pipeline() primește o serie de parametri ce constituie un lanț de procesare. Astfel se folosește pentru normalizarea datelor elementul numit StandardScaler. Urmează apoi definirea clasificatorului LinearSVC, care are de asemenea câțiva parametri configurabili cum ar fi de exemplu opțiunea de a blansa setul de date sau de a seta numărul maxim de iterații ce vor fi rulate înainte să fie oprită execuția, în caz că nu se atinge convergența.

După ce s-a rulat instrucțiunea prezentată mai sus, clasificatorul este inițializat și poate fi folosit pentru a modela datele.

6.4.3 Definire TabNet

Pentru acest experiment s-a folosit o implementare în Pytorch a rețelei TabNet, preluată de pe platforma Github. Pentru început se rulează o instrucțiune ce acceseaza aplicația de gestiune de pachete pip, pentru a instala fișierele modelului în proiect.

```
pip install pytorch-tabnet
```

Odată ce a fost instalat pachetul pytorch-tabnet, procesul de definire al modelului este destul de simplu, fiind vorba despre un constructor ce poate primi o serie de parametri care includ foarte multe aspecte cum ar fi de exemplu dimesiunea măștii, numărul de pași din rețea, algoritmul de optimizare, rata de învățare etc.

```
clf = TabNetClassifier(
    n_d=8, n_a=8, n_steps=9,
    gamma=1.5, n_independent=2, n_shared=2,
    lambda_sparse=1e-3, momentum=0.02,
    optimizer_fn=torch.optim.Adam,
    optimizer_params=dict(lr=1e-3),
    scheduler_params = {"gamma": 0.95,
                        "step_size": 20},
    scheduler_fn=torch.optim.lr_scheduler.StepLR, epsilon=1e-15
)
```

Pentru a inițializa clasificatorul cu configurația predefinită se poate rula instrucțiunea fără parametri.

```
clf = TabNetClassifier()
```

Dupa ce a fost rulată instrucțiunea de mai sus, modelul este setat și poate fi folosit în procesul de antrenare.

6.5 ANTRENARE

Următoarea etapă este destinată antrenării modelelor și diferă pentru fiecare experiment.

6.5.1 Antrenare MLP

Pentru început, se va rula o celulă pentru a seta dispozitivul pe care se realizează antrenarea. Pentru a putea paraleliza eficient, procesul de antrenare se va rula pe GPU, folosid platforma CUDA. Dacă sesiunea curentă nu dispune de o placă grafică, atunci dispozitivul principal pe care se vor efectua calculele va fi CPU-ul.

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

Ulterior modelul de rețea neurală va fi încărcat pe dispozitivul principal de calcul, care în acest caz este procesorul grafic.

```
net = Net().to(device)
```

Pentru rețeaua personalizată bazată pe modelul MLP, procesul de testare a fost configurat manual și presupune două bucle for. Una este folosită pentru a parcurge numărul de epoci destinate antrenării (stabilite anterior printr-o variabilă), iar cea de-a doua parcuge structura de tip DataLoader ce a fost prezentată mai sus. Se va inițializa de asemenea un vector pentru urmărirea parcursului funcției de cost și se va înregistra timpul de antrenare.

```
EPOCHS = 50
losses = []
t = time.time()

for epoch in range(EPOCHS):
  running_loss = 0.0

  for data in trainset:
    x , y = data
    x = x.to(device)
    y = y.to(device)

    # set the gradients to zero before backprop
    optimizer.zero_grad()

    # pass X through the net
    output = net(x.float())

    # set loss function
    loss = criterion(output, y)

    loss.backward()
```

```
    optimizer.step()
  print("Epoch " + str(epoch + 1) + ' loss: %.4f' % (loss))

elapsed = time.time() - t
```

Pentru fiecare element din DataLoader se vor trimite pe dispozitivul principal atât un număr de exemple egal cu dimensiunea lotului pentru antrenare cât și etichetele aferente. Aceasta este modalitatea de aprovizionare a rețelei cu date. Înainte de a trece exemplele prin rețea, sunt setați gradienții în zero, pentru a nu se acumula cu cei calculați anterior. Variabilei output îi sunt atribuite predicțiile rezultate din trecerea exemplelor prin rețea. O conversie a datelor în tipul float este necesară din motive de compatibilitate.

În continuare este folosită funcția de cost căreia îi sunt pasate predicțiile rețelei și etichetele adevărate ale datelor. Se poate începe procesul de propagare înapoi, calculându-se gradienții și rulându-se un pas al algoritmului de optimizare.

Acest proces se repetă până ce toate datele din setul de antrenare au trecut prin rețea, pentru un număr de ori egal cu numărul de epoci setat la început.

Odată ce s-a lansat în execuție celula care cuprinde instrucțiunile prezentate mai sus și aceasta rulează pentru numărul setat de epoci, modelul este antrenat și poate fi evaluat pe lotul de test.

6.5.2 Antrenare LinearSVC

Dată fiind natura de algoritm implementat într-o bibliotecă de învățare profundă, modelul LinearSVC poate fi antrenat rulând o singură instrucțiune. Metoda fit() este aplicată clasificatorului identificat prin referința clf. Aceasta primește la intrare vectorul cu exemple și vectorul cu etichetele aferente, toate in format numpy array, nemaifiind necesară vreo conversie sau alt mecanism de aprovizionare a datelor. Este de asemnea măsurat timpul de antrenare.

```
t = time.time()
clf.fit(np_X_train, np_Y_train)
elapsed = time.time() - t
```

După rularea instrucțiunii prezentate mai sus, modelul este antrenat și se pot face predicții pe lotul de test.

6.5.3 Antrenare TabNet

Procesul de antrenare pentru experimentele cu rețeaua TabNet este foarte similar cu cel pentru LinearSVC, fiind nevoie de o singură instrucțiune. Diferența este consituită de parametrii pasați metodei fit(). În acest caz, se vor oferi funcției atât datele din lotul de antrenare cât și cele din lotul de test sau de validare, pentru optimizare. Vectorii pasați metodei sunt de tip numpy array, nefiind nevoie de o structură specială pentru aprovizionarea rețelei cu date. Pentru acestă rețea s-a realizat și o împarțire suplimentară a bazei de date, extrăgându-se un lot de validare din cel de antrenare. Mai multe detalii vor fi oferite în *Capitolul 7*.

```
clf.fit(X_train, Y_train, X_valid, Y_valid)
```

Similar cu experimentul LinearSVC, odata ce a fost rulată instrucțiunea de mai sus, se poate evalua modelul.

6.6 EVALUAREA PERFORMANȚELOR

Modulul responsabil cu evaluarea este de asemenea implementat diferit pentru fiecare experiment, însă funcționalitatea sa este aceeași. Se vor folosi modelele antrenate pe lotul de antrenare pentru a realiza clasificări pe exemple din lotul de test. Performanțele modelelor sunt de cele mai multe ori exprimate în măsuri de forma acurateței pe lotul de test sau pe lotul de antrenare.

6.6.1 Evaluare MLP

Pentru experimentul cu MLP, celula de testare se aseamănă cu cea de antrenare, însă de data aceasta se folosește o metodă din biblioteca Pytorch care specifică că nu se relizează antrenare. Se parcurge de asemnea lotul de test folosind un element de tip DataLoader, până cand s-au parcurs toate exemplele. Folosind contori, la final se afișează valoarea finală a acurateței în procente.

```
correct = 0
total = 0

with torch.no_grad():
  for data in testset:

    x, labels = data
    x = x.to(device)
    labels = labels.to(device)

    outputs = net(x.float())
    _, predicted = torch.max(outputs.data, 1)
    print(predicted)
    total += labels.size(0)
    correct += (predicted == labels).sum().item()

print('Accuracy of the network on the testset: %f %%' % (100 * correct /
total))
```

Dacă se dorește rularea modulului de evaluare pentru a verifica acuratețea pe lotul de antrenare, se poate înlocui elementul de tip DataLoader cu cel corespondent setului de antrenare.

6.6.2 Evaluare LinearSVC

Pentru modelul care folosește clasificatorul LinearSVC implementat în biblioteca Scikit-learn, procesul de evaluare este unul foarte simplu, fiind nevoie doar de o singură metodă de nivel înalt de abstractizare, căreia i se pasează vectori cu exemple din lotul de test, precum și etichetele aferente, direct în format numpy array. Astfel, pentru a evalua performațele modelului în termeni de acurateț pe lotul de test se vor rula următoarele instrucțiuni.

```
# Print Test Accuracy Score
print('Test Accuracy Score: ')
print(clf.score(np_X_test, np_Y_test))
```

Dacă se dorește testarea pe lotul de antrenare, se vor înlocui vectorii pasați ca parametri funcției score() cu cei aferenți setului de antrenare.

6.6.3 Evaluare TabNet

În cazul experimentelor cu TabNet, evaluarea se realizează similar cu cea pentru clasificatorul LinearSVC, fiind nevoie de doar o singură instrucțiune pentru a se clacula acuratețea pe lotul de test. Se va folosi de asemenea și o variabilă de tip buffer, care să înmagazineze predicțiile realizate de rețea cu ajutrul metodei predict() aplicată clasificatorului. Funcția accuracy_score() primește ca parametri de intrare predicțiile făcute de model și adevăratele etichete ale datelor și calculează acuratețea.

```
y_pred_test = clf.predict(X_test)
test_acc = accuracy_score(y_pred=y_pred_test, y_true=Y_test)
print('Test accuracy: ' +  str(test_acc))
```

Similar cu celelate experimente, dacă se dorește aflarea rezultatelor pe lotul de antrenare, se încoluiesc intrările funcției cu buffer-ul în care se stochează predicțiile făcute pe lotul de antrenare și etichetele adevarate aferente.

6.7 SALVAREA ȘI ÎNCĂRCAREA

Celulele de salvare și încarcare sunt opționale și nu afectează funcționarea rețelelor, dar pot fi utile atunci cand se dorește reantrenarea unui model în mai multe sesiuni, fară a se pierde progresul.

Folosind, de exemplu în cazul experimentelor cu MLP, o cale unde se vor salva respectiv de unde se vor încărca dicționarele cu paramertri, procesele de încărcare și salvare devin foarte simple. Implementarea este realizată foloisnd funcții din biblioteca Pytorch.

Pentru salvare:

```
save_path = '/content/drive/My Drive/Colab Notebooks/saved_models/model_N
ame'
torch.save(net.state_dict(), save_path)
```

Respectiv pentru încărcare:

```
net = Net()
net.load_state_dict(torch.load(save_path))
```

6.8 REPREZENTAREA REZULTATELOR

Sunt prezente și celule cu ajutorul cărora se pot vizualiza mai bine datele, prin intermediul graficelor, rapoartelor de clasificare sau a matricelor de confuzie.

Pentru exemplificare se vor prezenta câteva astfel de celule folosite în experimente. Aceste celule sunt opționale și implementarea lor diferă de la experiment la experiment, dar de principiu toate folosesc funcții predefinite din bibliotecile de învățare automată prezentate anterior cât și din biblioteci uzuale Python cum ar fi matplotlib.

Pentru reprezentarea sub formă de grafic a evoluției funcției de pierdere atât pe lotul de antrenare cât și pe cel de validare se folosește funcția plot() căreia i se pasează vectori ce conțin valorile în secvență a măsurilor de interes.

```
blue_patch = mpatches.Patch(color='blue', label='lot de antrenare')
red_patch = mpatches.Patch(color='red', label='lot de validare')
plt.legend(handles=[blue_patch, red_patch])
plt.grid()
plt.plot(clf.history['train']['loss'], 'b')
```

```
plt.plot(clf.history['valid']['loss'], 'r')
```

Similar se foloește și pentru evoluția acurateței.

```
blue_patch = mpatches.Patch(color='blue', label='lot de antrenare')
red_patch = mpatches.Patch(color='red', label='lot de validare')
plt.legend(handles=[blue_patch, red_patch])
plt.grid()
plt.plot([-x for x in clf.history['train']['metric']], 'b')
plt.plot([-x for x in clf.history['valid']['metric']], 'r')
```

Pentru afișarea unei matrice de confuzie se pot folosi mai multe metode, utilizând funcții din diferite biblioteci cum ar fi seaborn, matplotlib sau Scikit-learn. Următoarele instrucțiuni generează o matrice de confuzie de tipul heatmap folosind o structură de tip DataFrame din biblioteca pandas.

```
confusion_matrix_df = pd.DataFrame(100 * confusion_matrix(Y_test, y_pred_
test, normalize="true"))
sns.heatmap(confusion_matrix_df, annot=True)
```

Pentru o implementare cu biblioteca matplotlib se folosesc funcții precum plot_confusion_matrix(), existând și posibilitatea de normalizare manuală a valorilor. Se pasează un vector cu numele claselor împreună cu clasificatorul și datele pe care se dorește a fi realizată reprezentarea, în cazul proiectului actual vectorii cu date de test și etichetele corespunzătoare.

```
clases = ['colics', 'discomfort', 'eructation', 'hunger', 'pain', 'tiredn
ess']

titles_options = [("Confusion matrix, without normalization", None),
                  ("Normalized confusion matrix", 'true')]
for title, normalize in titles_options:
    disp = plot_confusion_matrix(clf, np_X_test, np_Y_test,
                                 display_labels=clases,
                                 cmap=plt.cm.Blues,
                                 normalize=normalize)
    disp.ax_.set_title(title)

    print(title)
    print(disp.confusion_matrix)

plt.show()
```

Pentru realizarea unui raport de clasificare, se apelează din nou la funcții predefinite, procedeul fiind unul foarte simplu.

```
print(classification_report(Y_test, y_pred_test))
```

CAPITOLUL 7

EXPERIMENTE

7.1 CONFIGURAȚIA EXPERIMENTALĂ

7.1.1 Setul de date

Pentru antrenare a fost folosită baza de date în format CSV extrasă din baza de data SPLANN în format WAV pusă la dispoziție de către laboratorul SpeeD. Baza de date rezultată este formată din 12939 de exemple etichetate repartizate în 6 clase, după nevoia pentru care sunt reprezentative.

Aceste date fost împărțite în loturi de antrenare și test respectând o proporție de 10% din totalul de exemple pentru setul de test și restul pentru antrenare. Proporția împărțirii este relativă la numărul de exemple al fiecărei clase. În principal, distribuția s-a realizat astfel încât exemple generate de același bebeluș să nu fie prezente în amblele loturi. Excepție de la acest caz face clasa reprezentativă pentru nevoia de oboseală, pentru care au fost generate probe de către un singur subiect și pentru care au fost alese aleator un procent de aproximativ 10% din exemple, pentru a respecta proporțiile folosite și la celelalte clase. Distribuția seturilor de date pentru antrenare și test este prezentată în tabelele 7.1 și 7.2.

Numărul clasei	Tipul clasei	Numărul de exemple de antrenare	Procentaj din totalul de exemple de antrenare
0	colici	225	1.93%
1	disconfort	2001	17.17%
2	eructație	455	3.90%
3	foame	4975	42.71%
4	durere	3962	34.01%
5	oboseală	30	0.25%

Tabel 7.1 - Distribuția datelor din lotul de antrenare

Numărul clasei	Tipul clasei	Numărul de exemple de test	Procentaj din totalul de exemple de test
0	colici	25	1.93%
1	disconfort	209	16.18%
2	eructație	50	3.87%
3	foame	561	43.45%
4	durere	442	34.23%
5	oboseală	4	0.30%

Tabel 7.2 - Distribuția datelor din lotul de test

Este de menționat faptul că pentru experimentele cu rețeaua TabNet au fost realizată o împarțire a bazei de date suplimentară în cod, dată fiind natura clasificatorului. Acesta folosește un set de validare pentru optimizarea modelului, și antrenarea folosind setul de test implică o interacțiune nedorită cu datele de test, schimbând rezultatele atunci când se realizează procesul de evaluare pe setul de test. În acest sens, s-au ales aleator doar din setul de antrenare un procent de 10% din exemple pentru a deservi drept set de validare. Alegerea exemplelor s-a realizat în mod proporțional cu numărul de exemple disponibil în fiecare clasă.

7.1.2 Resurse hardware și software

Cu privire la resursele software, procesul de extragere a trăsăturilor s-a realizat folosind extractorul de trăsături openSMILE în versiunea cu interfață grafică pentru procesarea pe loturi, pe o mașină cu sistem de operare Windows. Dezvoltarea sistemelor a avut loc pe platforma Google Colab, folosind un browser, limbajul Python 3, diverse biblioteci uzuale Python precum și bibliotecile de învățare automată Pytorch, Skicit-learn și în ultimul rând o impementare în Pytorch a rețelei TabNet, preluată de pe platforma Github.

În ceea ce privește resursele hardware, toate experimentele au beneficiat de resurse generoase oferite de platforma Google Colab, atât în ceea ce privește memoria RAM cât și accesul la procesoare grafice prin intermediul platformei CUDA. Deși componentele explicite pe care se rulează aplicațiile sunt transparente utilizatorului, fiind alocate adaptiv, platforma Google Colab

oferă informații despre o suită de procesoare grafice care pot fi folosite în procesul de antrenare. Acestea includ plăci video de tipul NVIDIA K80, P100, P4, T4, și V100 [33].

7.2 REZULTATE

7.2.1 Rezultate MLP

Pentru început s-a experimentat cu o rețea de tip MLP cu 4 straturi complet conectate. Structura de bază a acesteia este:

(fc1): Linear(in_features=6373, out_features=1024, bias=True)

(fc2): Linear(in_features=1024, out_features=256, bias=True)

(fc3): Linear(in_features=256, out_features=64, bias=True)

(fc4): Linear(in_features=64, out_features=6, bias=True)

S-a optat pentru funcții de activare de tip ReLu, rolul de funcție de cost este ocupat de entropia încrucișată iar algoritmul de optimizare este Adam. Nu au fost folosite măsuri de regularizare suplimnetare. Acest model a fost antrenat pentru 50 de epoci, pentru mai multe valori ale ratei de învățare. Rezultatele obținute sunt expuse în tabelul 7.3. Este de menționat că în acest timp, antrenarea se realizează pe GPU.

Rata de învățare	Acurateță pe lotul de antrenare	Acurateță pe lotul de test	Timp de antrenare pentru 50 de epoci
1E-03	42.71%	43.45 %	135s
5E-04	42.71 %	43.45 %	95s
1E-04	34.04 %	34.23 %	96s
5E-05	34.06%	34.23 %	95s
1E-05	40.19 %	40.66 %	95s

Tabel 7.3 - Rezultate MLP 4 straturi, pentru 50 de epoci

Observând rezulatele obținute, s-a decis antrenarea modelului pentru mai multe epoci (200), pentru a obține o imagine mai clară asupra abilității de generalizare a rețelei. S-au repetat experimentele anterioare în ceea ce privește variația ratei de învățare și au rezultat datele din tabelul 7.4.

Rata de învățare	Acurateță pe lotul de antrenare	Acurateță pe lotul de test	Timp de antrenare pentru 50 de epoci
1E-03	42.71 %	43.45 %	380s
5E-04	42.71 %	43.45 %	382s
1E-04	42.71 %	43.45 %	380s

5E-05	42.71 %	43.45 %	382s
1E-05	42.85 %	43.45 %	387s

Tabel 7.4 - Rezultate MLP 4 straturi, pentru 200 de epoci

Având aceste informații, s-a observat faptul că acuratețea maxima obținută în ambele cazuri nu depășea valoarea aproximativă de 43%, iar in celelate cazuri era de aproximativ 34%. Aceste cifre au fost corelate cu procentele de apariție ale claselor cele mai voluminoase ca și număr de exemple și astfel s-a teoretizat că modelul întâmpină fenomenul de supraantrenare. Odată ce s-au analizat ieșirile rețelei, teoria a fost confirmată. S-a determinat că în funcție de statulul aleator în care pornește rețeaua aceasta se blochează pe una dinte cele două clase cele mai numeroase și nu mai învață, scoțând la ieșire doar eticheta clasei pe care a supraîncadrat-o.

Prima încercare de evitare a acestui fenomen a constat în modificarea arhitecturii. Pentru a reduce numărul de parametri și a simplifica rețeaua, s-a ales următoarea structură cu doar 3 straturi:

(fc1): Linear(in_features=6373, out_features=256, bias=True)

(fc2): Linear(in_features=256, out_features=256, bias=True)

(fc3): Linear(in_features=256, out_features=6, bias=True)

S-au păstrat aceleași alegeri pentru funcții de activare, funcție de cost și algoritm de optimizare. Din păcate, această micșorare de rețea nu a fost suficientă, obținându-se aceleași scoruri de acuratețe. O metodă de regularizare încercată apoi a fost balasarea setului de antrenare cu ponderi invers proporționale cu rata de apariție a exemplelor din fiecare clasă. Rezultatele cu și fără balansare pot fi observate în tabelul 7.5.

Set de date balansat	Rată de învățare	Acuratețe pe lotul de antrenare	Acuratețe pe lotul de test
Nu	1E-03	42.71 %	43.45 %
Da	1E-03	42.62 %	43.37 %

Tabel 7.5 - Rezultate MLP 3 straturi cu și fără balansare

În final s-a recurs la folosirea metodei de regularizare cu renunțare la neuroni în diferite ponderi. Analizând ieșirile rețelei s-a constat că într-adevăr aceasta nu mai întâmpină fenomenul de supraantrenare, însă performanțele au rămas în continuare destul de scăzute. Acuratețea maximă atinsă fără configurări suplimentare a fost de 38.77%.

Ca o primă concluzie, se poate afirma că utlizarea unei arhitecturi bazată pe modelul perceptronului multistrat nu este prea eficientă în cazul sarcinii de față. Posibile motive pentru aceast neajuns ar putea fi gradul înalt de dimensionalitate a datelor de intrare, transformând problema de clasificare într-una foarte complexă.

7.2.2 Rezultate LinearSVC

Pentru experimentul cu LinearSVC s-a urmărit în principal analiza comportamentului unui clasificator liniar pe setul de date aflat la dispoziție. S-a experimentat mai întâi cu o combinație de două clase, pentru a verifica buna funcționare a modelului. Ulterior s-a mai adăugat o clasă, pentru a observa influența complicării sarcinii de clasificare asupra performanțelor modelului, în termeni de acuratețe. După experimentul cu 3 clase s-a încercat și cu 4, pentru avea o imagine mai clară asupra felului în care se comportă modelul cu mai multe clase. Indicii corespunzători fiecărei

clasă sunt expuși în tabelul 7.6, iar rezultatele experimentelor menționate mai sus sunt prezentate în tabelul 7.7. În acest tabel apar și valorile pentru parametri care au fost variați mai târziu.

Indicele clasei	Tipul clasei
0	colici
1	disconfort
2	eructație
3	foame
4	durere
5	oboseală

Tabel 7.6 - Indici clase

Combinație de clase	C	max_iter	tol	Acuratețe pe lotul de antrenare
0 - 1	1	1000	1E-05	96.16 %
0 - 1 - 2	1	1000	1E-05	71.83 %
0 - 1 - 2 - 3	1	1000	1E-05	46.39 %

Tabel 7.7 - Rezultate creștere succesivă a complexității

Deja de la introducerea celei de-a patra clase s-a observat o scădere drastică a scorului pe lotul de test și odată cu aceasta, modelul nu a reușit să mai atingă convergența. Pentru a solidifica aceste observații, s-a încercat antrenarea unui model cu 5 clase, pentru a se analiza impactul asupra performanțelor și pentru a se determina dacă se poate atinge convergența variind parametri modelului. Rezultatele obținute în cadrul acestui experiment sunt expuse în tabelul 7.8.

Combinație de clase	C	max_iter	tol	Acuratețe pe lotul de test	Acuratețe pe setul de antrenare	Timp de antrenare
0 - 1 - 2 - 3 - 4	1	1000	1E-05	38.92 %	99.88 %	396s
	1	2000	1E-05	38.69 %	99.91 %	634s
	1	3000	1E-05	39.08%	99.90%	844s
	0.8	3000	1E-04	39.00 %	99.93 %	859s
	0.7	3000	1E-04	39.00 %	99.90%	846s
	100	3000	1E-04	39.23 %	99.86 %	854s
	1000	3000	1E-04	39.54 %	99.87 %	824s
	1000	5000	1E-04	39.54 %	99.87 %	1285s

Tabel 7.8 - Rezultate LinearSVC cu 5 clase

Deși au fost variați mai mulți parametri, niciun model nu a reușit să atingă convergența, iar performanțele au rămas în continuare joase. Acest fapt nu este deloc surprinzător, dată fiind atât natura liniară a clasificatorului cât și complexitatea datelor.

Pornind de la aceste observații, s-a decis experimentarea sarcinii de clasificare binară folosind toate combinațiile posibile de clase, pentru a determina măsura în care acestea sunt separabile liniar două câte două. Experimentele au generat rezultatele din tabelul 7.9.

Combinație de clase	C	max_iter	tol	Acurateță pe lotul de test	Acurateță pe lotul de antrenare	Timp de antrenare	Note
0 - 1	1	1000	1E-04	96.15%	100%	6s	
0 - 2	1	1000	1E-04	76%	100%	0s	
0 - 3	1	1000	1E-04	95.90%	100%	22s	
0 - 4	1	1000	1E-04	97.64%	100%	18s	
0 - 5	1	1000	1E-04	100%	100%	0s	
1 - 2	1	1000	1E-04	71.42%	100%	4s	
1 - 3	1	1000	1E-04	52.85%	99.98%	57s	Nu a atins convergența
1 - 3	1	2000	1E-04	52.85 %	100%	108s	
1 - 4	1	1000	1E-04	54.99 %	100%	49s	Nu a atins convergența
1 - 4	1	2000	1E-04	55.14%	100%	69s	
1 - 5	1	1000	1E-04	97.18%	100%	33s	Nu a atins convergența
2 - 3	1	1000	1E-04	87.23%	100%	18s	
2 - 4	1	1000	1E-04	89.02%	100%	13s	
2 - 5	1	1000	1E-04	51.85%	100%	1s	
3 - 4	1	1000	1E-04	66%	100%	69s	Nu a atins convergența
3 - 4	1	2000	1E-04	65.90%	100%	127s	Nu a atins convergența
3 - 5	1	1000	1E-04	98.76%	100%	30s	
4 - 5	1	1000	1E-04	99.10%	100%	29s	

Tabel 7.9 - Rezutate LinearSVC, combinații de 2 clase

Aceste informații au fost mai apoi puse sub formă de matrice în tabelul 7.10 pentru a se evidenția care clase sunt cele mai similare precum și care clase sunt cel mai ușor separabile liniar. Această formă de vizualizare relevă faptul că între anumite nevoi cum ar fi de exemplu disconfortul și foamea se discerne foarte greu, la fel și pentru durere și foame. Între eructație și oboseală, există un grad înalt de similaritate, însă aceste rezultate nu sunt foarte conclusive deoarece numărul de

exemple pentru nevoia de oboseală este foarte mic. Pentru celelalte combinații de clase, abiliatea de diferențiere a clasificatorului LinearSVC este decentă, obținându-se performanțe mai bune.

Clasa	0 (colici)	1 (disconfort)	2 (eructație)	3 (foame)	4 (durere)	5 (oboseală)
0 (colici)		96.15%	76%	95,90%	97,64%	100%
1 (disconfort)	96.15%		71,42%	52,85%	54,99 %	97.18%
2 (eructație)	76%	71,42%		87,23%	89,02%	51,85%
3 (foame)	95,90%	52,85%	87,23%		66%	98.76%
4 (durere)	97,64%	54,99 %	89,02%	66%		99,10%
5 (oboseală)	100%	97.18%	51,85%	98.76%	99,10%	

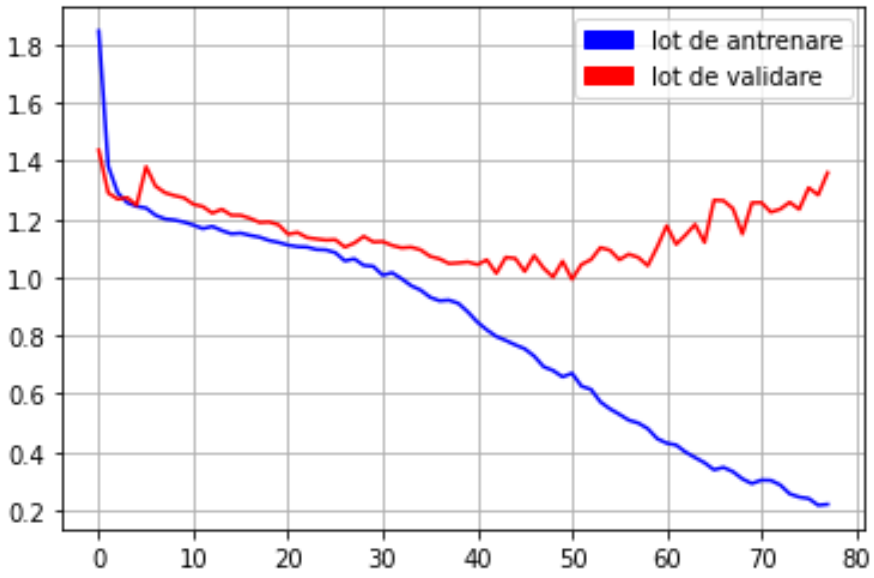
Tabel 7.10 - Rezultate combinație de 2 clase, formă matricială

Ca și concluzie preliminară, informațiile obținute în urma acestor experimente consolidează teoria menționată mai devreme, conform căreia gradul înalt de dimensionalitate a datelor face ca transformările liniare să nu fie neapărat eficiente în această sarcină. De asemenea, aceste experimente au relevat măsura avansată de similaritate pe care anumite clase o au, făcând abordarea acuală și mai puțin viabilă.

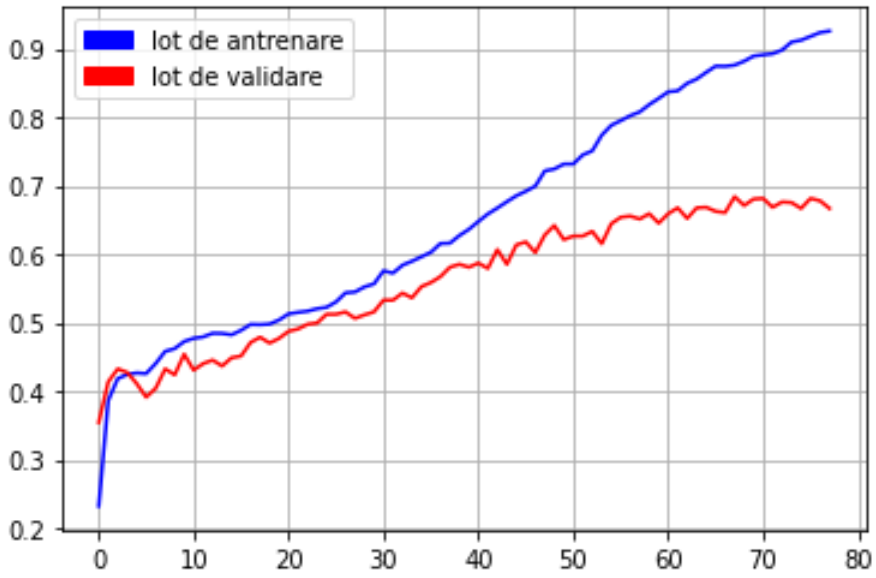
7.2.3 Rezultate TabNet

Experientele cu TabNet au folosit pentru început aceeași împărțire a setului de date precum celelalte experimente prezentate anterior. S-a obținut o acuratețe de 44% pe lotul de test, însă acest rezultat nu este în totalitate comparabil cu cele obținute în lucrările anterioare de cercetare, dată fiind modalitatea în care se face antrenarea. Clasificatorul rețelei TabNet primește la intrare și parametri pentru optimizare, și anume un lot de validare. Antrenarea cu setul de date de test folosit drept set de validare nu oferă rezultate utile, doarece astfel modelul are acces la informații ale setului de test în timpul antrenării și nu se mai obține ulterior o evaluare corectă.

După ce s-a realizat împărțirea cu lot de validare provenit din lotul de antrenare, rezultatele sunt valide în contextul comparației cu celelate lucrări de cercetare ce tratează același subiect. Următorarele figuri expun performanțele rețelei TabNet, folosind setul de validare în procesul de antrenare.



Figură 7.1 - Evoluția valorilor funcției de cost pe epoci

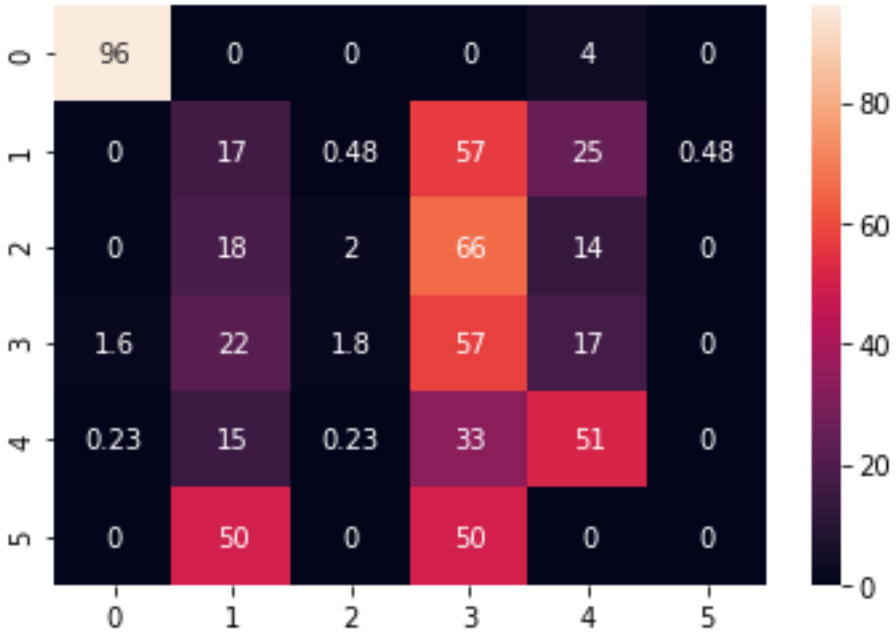


Figură 7.2 - Evoluția acurateței pe epoci

	precision	recall	f1-score	support
0.0	0.71	0.96	0.81	25
1.0	0.15	0.17	0.16	209
2.0	0.08	0.02	0.03	50
3.0	0.52	0.57	0.54	561
4.0	0.59	0.51	0.55	442
5.0	0.00	0.00	0.00	4
accuracy			0.47	1291
macro avg	0.34	0.37	0.35	1291
weighted avg	0.47	0.47	0.47	1291

Figură 7.3 - Raport de clasificare

Pentru matricea de confuzie, etichetele claselor sunt aceleași ca pentru experimentele cu LinearSVC.



Figură 7.4 - Matrice de confuzie

În final s-a obținut o acuratețe de 47% pe lotul de test, folosind configurația prestabilită a modelului TabNet. Acest rezultat este promițător, fiind cel mai bun scor obținut până acum în experimentele din cadrul acestui proiect. Dată fiind structura tabulară a bazei de date cu trăsături extrase, se observă o performanță mai bună a rețelei TabNet decât a celorlalți clasificatori încercați pe abordarea propusă.

CAPITOLUL 8

CONCLUZII

8.1 CONCLUZII

Sarcina de clasificare de plânset de bebeluș este una destul de dificilă, însa în același timp ridică probleme interesante. Un sistem automat capabil să îndeplinească cu succes această sarcină ar fi de mare folos pentru părinții tineri, însă dezvoltarea acestuia se dovedește a fi o reală provocare. O implementare folosind rețele neurale profunde pare să fie o abordare bună, însă este dificil de realizat practic cu o rată bună de succes, dată fiind natura complexă a problemei.

Lucrarea de față și-a propus să consolideze cercetările anterioare din domeniu, încercând o abordare de rețea neurală bazată pe tehnci de învățare profundă. Aspectul principal de luat în considerare a fost structura bazei de date pe care s-a lucrat. Propunerea de a lucra pe date cu trăsături extrase a mai fost încercată însă nu cu tehnici de învățare profundă.

În procesul de dezvoltare al unui sistem de clasificare bazat pe rețele neurale intervin multe dificultăți, atât de natură teoretică cât și practică. Prin definiție, procesul de dezvoltare în domeniul inteligenței artificiale este unul de încercare și eroare, prin prisma faptului că nu există soluții care să funcționeze eficient pentru orice tip de sarcină și că în sine sarcinile abordate în acest domeniu au particularități vast diferite. Odată ce s-a ales un punct de pornire, se experimentează

incremental, cu schimbări de arhitecutră, cu variații de parametri, în scopul găsirii celei mai bune soluții.

Dată fiind natura complexă a subiectului acestui proiect, s-a încercat dezvoltarea unui sistem care să obțină rezultate mai bune decât cele obținute până acum, dar scopul principal a fost cel de cercetare. Datorită experimentelor efectuate, au ieșit la iveală informații noi și utile pentru viitoare cercetări.

Experimentele cu arhitectură de tip MLP au demonstrat faptul că un clasificator de acest fel este foarte vulnerabil la fenomenul de supraantrenare pe o bază de date de tipul celei folosite în cadrul acestui proiect. De asemnea, s-a concretizat ideea că datele cu o dimensionalitate de grad înalt transformă sarcina de clasificare într-una foarte complexă, fiind și mai aparentă nevoia selecției de trăsături cu impact semnificativ.

Investigațiile realizate cu clasificatorul LinearSVC au relevat gradul de similaritate între anumite clase din cadrul bazei de date SPLANN în format de trăsături extrase, precum și măsura în care acestea sunt separabile liniar. Informațiile acestea pot fi folosite mai departe în dezvoltarea altor categorii de clasificatori.

Experimentele realizate folosind rețeaua TabNet au costituit de asemena o sursă de informații valoroase. Rezultatele obținute de acest model au fost cele mai bune din cadrul proiectului, deschizând calea spre cercetări ulterioare care să profite de natura tabulară a setului de date cu trăsături extrase.

8.2 CONTRIBUȚII PERSONALE

Punctul de pornire pentru acest proiect a fost generarea bazei de date în format CSV, prin extragere de trăsături, pornind de la baza de date SPLANN, pusă la dispoziție de laboratorul SpeeD.

Printre contribuțiile personale aduse acestui proiect se număra:

- Generarea bazei de date formată din trăsături extrase;
- Dezvoltarea modulelor constituente ale sistemelor, atât în ceea ce privește scrierea de cod pentru încărcarea, preprocesarea și formatarea datelor, cât și construirea și configurarea rețelelor respectiv implementrarea etapelor de antrenare și evaluare;
- Conceperea și efectuarea experimentelor diferite pentru fiecare tip de clasificator;
- Interpretarea rezultatelor obținute;

8.3 POSIBILE DEZVOLTĂRI ULTERIOARE

Câteva posibile direcții de cercetare ulterioară includ:

- Aplicarea unui procedeu de optimizare automată a parametrilor rețelei TabNet, în scopul obținerii unei performanțe mai bune, date fiind rezultatele promițătoare obținute de aceasta;
- Implementarea unui mecanism capabil să realizeze o selecție a celor mai bune trăsături, reducând astfel gradul ridicat de dimensionalitate al datelor, cu posibilitatea de îmbunătățire a performațelor modelelor de tip MLP;
- Încercarea unei abordări diferite care folosește tehnici de învățare profundă, spre exemplu folosirea unei rețele neurale convoluționale, care lucrează pe spectrograme extrase din fișierele audio;

REFERINȚE

[1] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Dunstan_Baby_Language. [Accessed 20 August 2020].

[2] M. Rusu, S. Diaconescu, G. Sardescu and E. Bratila, "Database and system design for data collection of crying related to infant's needs and diseases," 2015.

[3] "SOFTWIN Research," [Online]. Available: http://www.softwinresearch.ro/index.php/en/research-projects/splann. [Accessed 22 August 2020].

[4] I.-A. Bănică, H. Cucu, A. Buzo, D. Burileanu and C. Burileanu, "Baby Cry Recognition in Real-World Conditions," 2016.

[5] R. I. TUDUCE, M. S. RUSU, H. CUCU and C. BURILEANU, "Automated Baby Cry Classification on a Hospital-acquired Baby Cry Database," 2019.

[6] B. Schuller and A. Batliner, "compare.openaudio.eu," 2019. [Online]. Available: http://www.compare.openaudio.eu/. [Accessed 23 August 2020].

[7] B. Schuller, S. Steidl, A. Batliner, A. Vinciarelli, K. Scherer, F. Ringeval, M. Chetouani, F. Weninger, F. Eyben, E. Marchi, M. Mortillaro, H. Salamin, A. Polychroniou, F. Valente and S. Kim, "The INTERSPEECH 2013 Computational Paralinguistics Challenge: Social Signals, Conict, Emotion, Autism," 2013.

[8] F. Ringeval, "Researchgate," [Online]. Available: https://www.researchgate.net/figure/COMPARE-acoustic-feature-set-65-provided-low-level-descriptors-LLD_tbl1_259889707. [Accessed 23 August 2020].

[9] A. Ng, "Coursera - Neural Networks and Deep Learning," [Online]. Available: https://www.coursera.org/learn/neural-networks-deep-learning/home/week/2. [Accessed 23 August 2020].

[10] J. B. Ahire, "The Artificial Neural Networks handbook: Part 1," 24 August 2018. [Online]. Available: https://medium.com/coinmonks/the-artificial-neural-networks-handbook-part-1-f9ceb0e376b4. [Accessed 23 August 2020].

[11] M. Hacibeyoglu, "Researchgate," [Online]. Available: https://www.researchgate.net/figure/The-basic-structure-of-convolution-neural-networks_fig1_328405250. [Accessed 23 August 2020].

[12] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Convolutional_neural_network#Neural_abstraction_pyramid. [Accessed 23 August 2020].

[13] "CS231n: Convolutional Neural Networks for Visual Recognition," [Online]. Available: http://cs231n.stanford.edu/. [Accessed 23 August 2020].

[14] "CS231n Convolutional Neural Networks for Visual Recognition," [Online]. Available: https://cs231n.github.io/convolutional-networks/. [Accessed 23 August 2020].

[15] M. Stewart, "Topbots - Advanced Topics in Deep Convolutional Neural Networks," 25 July 2019. [Online]. Available: https://www.topbots.com/advanced-topics-deep-convolutional-neural-networks/. [Accessed 24 August 2020].

[16] D. Radečić, "Towards Data Science - Softmax Activation Function Explained," 18 June 2020. [Online]. Available: https://towardsdatascience.com/softmax-activation-function-explained-a7e1bc3ad60. [Accessed 25 August 2020].

[17] R. Jain, "Hackerearth - 3 Types of Gradient Descent Algorithms for Small & Large Data Sets," 7 March 2017. [Online]. Available: https://www.hackerearth.com/blog/developers/3-types-gradient-descent-algorithms-small-large-data-sets/. [Accessed 25 August 2020].

[18] V. Bushaev, "Towards Data Science - Stochastic Gradient Descent with momentum," 4 Decembrie 2014. [Online]. Available: https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d. [Accessed 25 August 2020].

[19] "CS231n Convolutional Neural Networks for Visual Recognition," [Online]. Available: https://cs231n.github.io/neural-networks-3/. [Accessed 25 August 2020].

[20] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Vanishing_gradient_problem. [Accessed 26 August 2020].

[21] B. Dickson, "TechTalks - The limits and challenges of deep learning," 2018 February 2018. [Online]. Available: https://bdtechtalks.com/2018/02/27/limits-challenges-deep-learning-gary-marcus/. [Accessed 26 August 2020].

[22] A. Gandhi, "Nanonets - Data Augmentation | How to use Deep Learning when you have Limited Data—Part 2," 2018. [Online]. Available: https://nanonets.com/blog/data-

augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/. [Accessed 27 August 2020].

[23] "JavaTpoint," [Online]. Available: https://www.javatpoint.com/python-features. [Accessed 23 August 2020].

[24] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/PyTorch. [Accessed 24 August 2020].

[25] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Scikit-learn. [Accessed 24 August 2020].

[26] J. Brownlee, "Machine Learning Mastery - A Gentle Introduction to Scikit-Learn: A Python Machine Learning Library," 16 August 2014. [Online]. Available: https://machinelearningmastery.com/a-gentle-introduction-to-scikit-learn-a-python-machine-learning-library/. [Accessed 24 August 2020].

[27] "Wikipedia," [Online]. Available: https://ro.wikipedia.org/wiki/CUDA. [Accessed 28 August 2020].

[28] R. Keim, "All About Circuits - How to Train a Multilayer Perceptron Neural Network," 26 December 2019. [Online]. Available: https://www.allaboutcircuits.com/technical-articles/how-to-train-a-multilayer-perceptron-neural-network/. [Accessed 29 August 2020].

[29] R. Pupale, "Towards Data Science - Support Vector Machines(SVM) — An Overview," 16 June 2018. [Online]. Available: https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989. [Accessed 29 August 2020].

[30] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Support_vector_machine. [Accessed 30 August 2020].

[31] P. Tune, "Towards Data Science - The Unreasonable Ineffectiveness of Deep Learning on Tabular Data," 26 April 2020. [Online]. Available: https://towardsdatascience.com/the-unreasonable-ineffectiveness-of-deep-learning-on-tabular-data-fd784ea29c33. [Accessed 30 August 2020].

[32] S. O. Arık and T. Pfister, "TabNet: Attentive Interpretable Tabular Learning," 2020.

[33] "Google Cloud," [Online]. Available: https://cloud.google.com/gpu. [Accessed 3 September 2020].

ANEXA 1

```
# Implementation using custom MPL

# Connection to Google Drive for CSV access
"""

from google.colab import drive
drive.mount('/content/drive')

"""# Imports cell"""

import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
import pandas as pd
import time
import gc

"""# CUDA implemetation, so we can run on GPU"""

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
# Assuming that we are on a CUDA machine, this should print a CUDA device:
print(device)

"""# Script for loading data"""

path = '/content/drive/My Drive/Colab Notebooks/extracted_features_csv/'
```

```
def loadWith(shuffle):
  train_data = np.empty([1, 6374])
  test_data = np.empty([1, 6374])
  train_data = np.delete(train_data, (0), axis=0)
  test_data = np.delete(test_data, (0), axis=0)

  def processAndAppend(cls, old_data):
    data = pd.read_csv(filename, delimiter=';', engine='c', dtype='a')
    data = data.to_numpy()
    data = data[:, 2:]
    label = np.zeros((data.shape[0], 1)) + cls
    data = np.append(data, label, axis=1)
    print("\tappended data shape = " + str(data.shape))
    if shuffle:
      np.random.shuffle(data)
    new_data = np.append(old_data, data, axis=0)
    return new_data

  clases = ['colics', 'discomfort', 'eructation', 'hunger', 'pain', 'tiredness']

  for cls in range(0, 5):
    print('Importing Class ' + str(cls) + '/5 examples...')
    filename = path + 'train_' + clases[cls] + '.csv'
    train_data = processAndAppend(cls=cls, old_data=train_data)
    filename = path + 'test_' + clases[cls] + '.csv'
    test_data = processAndAppend(cls=cls, old_data=test_data)
    print("\ttrain_data shape = " + str(train_data.shape))
    print("\ttest_data shape = " + str(test_data.shape))

  return train_data, test_data

print("Data import script loaded.")

"""# Import Data from CSVs into Numpy Arrays"""

print('Commencing data import:')
t = time.time()
np_train_data, np_test_data = loadWith(shuffle=False)
elapsed = time.time() - t
print('Data import finished!\nLoading took: ' + str(round(elapsed)) + ' seconds\n')

print('Final train_data shape: ' + str(np_train_data.shape))
print('Final test_data shape: ' + str(np_test_data.shape))

"""# Splitting data in Train and Test"""

np_X_train = np_train_data[:, :-1]
np_Y_train = np_train_data[:, -1]
np_X_test = np_test_data[:, :-1]
np_Y_test = np_test_data[:, -1]

print('X_train shape: ' + str(np_X_train.shape))
print('Y_train shape: ' + str(np_Y_train.shape))
print('X_test shape: ' + str(np_X_test.shape))
print('Y_test shape: ' + str(np_Y_test.shape))

"""# Data formatting"""

# Cast Numpy Arrays to float64 for pytorch Tensor conversion
```

```
np_X_train = np_X_train.astype(np.float64)
np_Y_train = np_Y_train.astype(np.float64)
np_X_test = np_X_test.astype(np.float64)
np_Y_test = np_Y_test.astype(np.float64)
print('Succesfully converted np_arrays to float64.')

X_trainTensor = torch.from_numpy(np_X_train)
Y_trainTensor = torch.from_numpy(np_Y_train)
X_testTensor = torch.from_numpy(np_X_test)
Y_testTensor = torch.from_numpy(np_Y_test)
print('Succesfully converted to Tensor objects.')

Y_trainTensor = Y_trainTensor.type(torch.LongTensor)
Y_testTensor= Y_testTensor.type(torch.LongTensor)
print('Succesfully converted to Long Tensor.')

train_data = torch.utils.data.TensorDataset(X_trainTensor, Y_trainTensor)
test_data = torch.utils.data.TensorDataset(X_testTensor, Y_testTensor)

trainset = torch.utils.data.DataLoader(train_data, batch_size=64, shuffle=True)
testset = torch.utils.data.DataLoader(test_data, batch_size=64, shuffle=True)
print('Succesfully built DataLoaders.')

"""# Clean memory of redundant data (Optional)"""

# Cleaning the memory of redundant information
# We only need the DataLoaders


del np_train_data
del np_test_data
del np_X_train
del np_Y_train
del np_X_test
del np_Y_test
del X_trainTensor
del Y_trainTensor
del X_testTensor
del Y_testTensor
del train_data
del test_data

gc.collect()

"""# Define the NN architecture"""

# Feed-forward MLP with 3 FC Layers

class Net(nn.Module):

  def __init__(self):
    super(Net, self).__init__()

    self.fc1 = nn.Linear(6373, 256)
    self.dropout1 = nn.Dropout(0.3)
    self.fc2 = nn.Linear(256, 256)
    self.dropout2 = nn.Dropout(0.2)
    self.fc3 = nn.Linear(256, 6)
    self.dropout3 = nn.Dropout(0.2)

  # x -> fc1/do1/relu -> out1 -> fc2/do2/relu -> out2 -> fc3/do3 -> ypred
```

```
  def forward(self, x):
    do1 = self.dropout1(self.fc1(x))
    out1 = F.relu(do1)

    do2 = self.dropout2(self.fc2(out1))
    out2 = F.relu(do2)

    do3 = self.dropout3(self.fc3(out2))
    y_pred = do3

    return y_pred.float()

# Initialize the NN and send it to GPU
net = Net().to(device)
print(net)

"""# Define loss and optimization functions"""

# specify loss function
# assign weights for the unballanced train set ( % of total for each class)
weights = [1/2.189, 1/17.178, 1/3.906, 1/42.711, 1/34.014, 1/0.257]
class_weights = torch.FloatTensor(weights).cuda()
criterion = nn.CrossEntropyLoss(weight=class_weights, reduction='mean')

# specify optimizer
optimizer = torch.optim.Adam(net.parameters(), lr=1e-3) # default: 1e-3

"""# Training stage"""

EPOCHS = 50
losses = []
t = time.time()

for epoch in range(EPOCHS):
  running_loss = 0.0

  for data in trainset:
    # data is a batch of featuresets and labels
    x , y = data
    x = x.to(device)
    y = y.to(device)

    # set the gradients to zero before backprop
    optimizer.zero_grad()

    # pass X through the net
    output = net(x.float())

    # set loss function
    loss = criterion(output, y)

    # do backprop
    loss.backward()
    optimizer.step()
  print("Epoch " + str(epoch + 1) + ' loss: %.4f' % (loss))

elapsed = time.time() - t
print('Training model for ' + str(EPOCHS) + ' Epochs finished!\nTraining took: ' +
str(round(elapsed)) + ' seconds\n')
```

```
"""# Save trained model"""

# Save trained model - change /name for each model
save_path = '/content/drive/My Drive/Colab Notebooks/saved_models/model_name
torch.save(net.state_dict(), save_path)
print("Model saved!")

"""# Load trained model"""

# Initialize just the path for loading only
save_path = '/content/drive/My Drive/Colab Notebooks/saved_models/model_name'

# Load trained model
net = Net()
net.load_state_dict(torch.load(save_path))
net = net.to(device)
print("Model loaded!")

"""# Testing stage"""

correct = 0
total = 0

with torch.no_grad():
  for data in testset:

    x, labels = data
    x = x.to(device)
    labels = labels.to(device)

    outputs = net(x.float())
    _, predicted = torch.max(outputs.data, 1)
    total += labels.size(0)
    correct += (predicted == labels).sum().item()

print('Accuracy of the network on the testset: %f %%' % (100 * correct / total))
```