Universitatea POLITEHNICA din București
Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Învățare automată pentru clasificarea particulelor de polen

Lucrare de licență

Prezentată ca cerință parțială pentru obținerea
titlului de *Inginer*
în domeniul *Calculatoare și Tehnologia Informației*
programul de studii *Ingineria informației*

Conducător științific
Conf. Dr. Ing. Horia CUCU
Ing. Mihai BOLDEANU

Absolvent
Gheorghe-Iulian CHIVU

Anul 2021

Universitatea "Politehnica" din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Program de studiu **INF**

Anexa 1

TEMA PROIECTULUI DE DIPLOMĂ

a studentului **CHIVU Gh. Gheorghe-Iulian , 442A**

1. Titlul temei: Învățare automată pentru clasificarea particulelor de polen

2. Descrierea temei și a contribuției personale a studentului (în afara părții de documentare):
###Descriere:
Proiectul are ca scop analiza celor mai puternici algoritmi de învătare automată ce pot fi utilizați cu succes în clasificarea automată a particulelor de polen care provin de la specii de plante diferite. Candidați viabili pentru o astfel de sarcină sunt algoritmi determiniști și algoritmi de clusterizare clasici/machine learning clasic (Perceptron, Naive Bayes, Nearest Neighbors, Gaussian Processes, MLP), respectiv algoritmi de învățare profundă precum rețelele convolutionale (CNN).
Analiza va fi realizată din mai multe perspective: complexitate algoritmi (necesită înțelegerea modului de funcționare a acestor algoritmi),
eficiență în timp (inferența se poate realiza în timp real sau doar offline?), respectiv acuratețea predicției.

###Contributii personale:
-Familiarizarea cu setul de date SAU_SRB
-Filtrarea și prelucrarea setului de date
-Aplicarea algoritmilor de preprocesare pe setul de date (PCA)
-Studiul algoritmilor de clusterizare clasici (Perceptron, Naive Bayes, Nearest Neighbors, Gaussian Processes, MLP)
-Studiul algoritmilor de DL ce pot fi utilizați pentru sarcina curentă (CNN)
-Adaptarea și aplicarea algoritmilor de segmentare pe setul de date SAU_SRB

###Cunoștințe necesare
-Programare: Python, pytorch
-DevOps: linux, gitlab

3. Discipline necesare pt. proiect:
PC, SDA, POO, PI, ISIA, RFIA

4. Data înregistrării temei: 2020-12-07 17:39:04

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

Student,
CHIVU Gh. Gheorghe-Iulian

Director departament,
Ș.L. dr. ing Bogdan FLOREA

Decan,
Prof. dr. ing. Mihnea UDREA

Cod Validare: **939c4acabe**

Declarație de onestitate academică

Prin prezenta declar că lucrarea cu titlul "*Învățare automată pentru clasificarea particulelor de polen*", 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 *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, *25.06.2021*

Absolvent *Gheorghe-Iulian CHIVU*

Cuprins

Lista figurilor 9

Lista tabelelor 10
Lista acronimelor 11

Introducere 12
Motivație 12
Obiective 12
Structura lucrării 13

1. Noțiuni teoretice/Metode propuse 14
1.1. Algoritmi de clasificare clasici (Machine learning) 14
1.1.1. Algoritmul Bayes naiv 14
1.1.2. Algoritmul celor mai apropiați vecini 14
1.1.3. Algoritmul proces gaussian 15
1.1.4. Perceptron 15
1.1.5. Perceptron multi strat 17
1.2. Algoritmi de învățare profundă 17
1.2.1. Rețele neurale convoluționale 17
1.2.2. Arhitectura LeNet-5 19
1.2.3. LeNet-5 aplicat pentru clasificare cifrelor MNIST 19

2. Clasificarea automată a polenului. Baza de date 20
2.1. Clasificarea automată a polenului 20
2.2. Descrierea bazei de date SAU-SRB 21
2.3. Preprocesarea datelor 22
2.3.1. Filtrarea și prelucrarea setului de date 23
2.3.2. Analiza componentelor principale 29

3. Clasificarea automată a polenului. Metode, arhitecturi și rezultate 30
3.1. Rezultate folosind metode clasice de clasificare 30
3.2. Clasificarea automată a polenului folosind rețele neurale convoluționale 34
3.2.1. Adaptarea LeNet-5 pentru clasificarea automată a polenului folosind imaginea de împrăștiere 34
3.2.2. Adaptarea LeNet-5 pentru clasificarea automată a polenului folosind spectrul de fluorescență 39
3.2.3. Adaptarea LeNet-5 pentru clasificarea automată a polenului folosind timpul de răspuns fluorescent 41
3.2.4. Arhitectura CNN multi-intrare 44
3.3. Sumar rezultate 46

Concluzii 47
Concluzii generale 47
Contribuții personale 47
Dezvoltări viitoare 48
Bibliografie 49

Lista figurilor

1.1. Rețea neurală de tip perceptron 15
1.2. Exemple de funcții de activare 16
1.3. Exemplu arhitectură MLP 17
1.4. Exemplu detector de contur folosind operația de convoluție 18
1.5. Arhitectura LeNet-5 19
1.6. Exemple eșantioane din baza de date MNIST 19
2.1. Vizualizarea celor 3 caracteristici din setul de date SAU-SRB 21
2.2. Vizualizarea a 15 imagini de împrăștiere din clasa Cynodon 22
2.3. Vizualizarea a 15 spectre de fluorescență din clasa Cynodon 23
2.4. Formatul inițial al unui eșantion din baza de date SAU-SRB 24
2.5. Formatul bazei de date prelucrate 25
2.6. Reprezentarea grafică a procesului de salvare a bazei de date prelucrată 26
2.7. Histogramele imaginilor de împrăștiere pentru fiecare clasă 27
2.8. Histogramele spectrelor de fluorescență pentru fiecare clasă 27
2.9. Histogramele timpilor de răspuns fluorescent pentru fiecare clasă 28
2.10. Histogramele celor 3 caracteristici pentru tot setul de date 28
3.1. Arhitectura MLP cu un singur strat ascuns 31
3.2. Arhitectura MLP cu multiple straturi ascunse 31
3.3. Reprezentarea arhitecturii LeNet-5 adaptată pentru imaginea de împrăștiere . . 35
3.4. Reprezentarea arhitecturii LeNet-5 modificată cu un strat convoluțional adițional pentru imaginea de împrăștiere 36
3.5. Reprezentarea arhitecturii LeNet-5 cu partea straturilor dense modificată pentru imaginea de împrăștiere 36
3.6. Vizualizarea funcției loss pentru antrenarea, respectiv testarea modelului folosind imaginea de împrăștiere 38
3.7. Reprezentarea arhitecturii LeNet-5 adaptată pentru spectrul de fluorecență . . . 39
3.8. Vizualizarea funcției loss pentru antrenarea, respectiv testarea modelului folosind doar spectrul de fluorescență 41
3.9. Reprezentarea arhitecturii LeNet-5 adaptată pentru timpul de răspuns fluorescent 42
3.10. Vizualizarea funcției loss pentru antrenarea, respectiv testarea modelului folosind doar timpul de răspuns fluorescent 43
3.11. Reprezentarea arhitecturii LeNet-5 multi-intrare bazată pe rețelele preantrenate și adaptată pentru cele 3 caracteristici 44
3.12. Reprezentarea arhitecturii LeNet-5 multi-intrare bazată pe arhitecturile precedente și adaptată pentru cele 3 caracteristici 45
3.13. Vizualizarea funcției loss pentru antrenarea, respectiv testarea arhitecturii multi-intrare 45

Lista tabelelor

2.1. Baza de date SAU-SRB neprelucrată 21
2.2. Statistica bazei de date SAU-SRB după aplicarea filtrării 24
3.1. Comparare rezultate inițiale cu rezultatele după aplicarea normalizării 32
3.2. Rezultate folosind algoritmi clasici de ML 33
3.3. Rezultate CNN pentru diverse dimensiuni ale filtrelor de convoluție folosind imaginea de împrăștiere 35
3.4. Rezultate CNN pentru diverse rate de antrenare folosind imaginea de împrăștiere 37
3.5. Rezultate CNN pentru diverse dimensiuni ale loturilor de antrenare folosind imaginea de împrăștiere 37
3.6. Rezultate după corectarea implementării algoritmului de învățare folosind imaginea de împrăștiere 38
3.7. Rezultate CNN pentru diverse dimensiuni ale filtrelor de convoluție folosind spectrul de fluorescență 39
3.8. Rezultate CNN pentru diverse rate de antrenare folosind spectrul de fluorescență 40
3.9. Rezultate CNN pentru diverse dimensiuni ale loturilor de antrenare folosind spectrul de fluorescență 40
3.10. Rezultate după corectarea implementării algoritmului de învățare folosind spectrul de fluorescență 40
3.11. Rezultate CNN pentru diverse dimensiuni ale filtrelor de convoluție folosind timpul de răspuns fluorescent 41
3.12. Rezultate CNN pentru diverse rate de antrenare folosind timpul de răspuns fluorescent 42
3.13. Rezultate CNN pentru diverse dimensiuni ale loturilor de antrenare folosind timpul de răspuns fluorescent 42
3.14. Rezultate după corectarea implementării algoritmului de învățare folosind timpul de răspuns fluorescent 43
3.15. Sumar rezultate 46

Lista acronimelor

CNN = Convolutional Neural Network
DL = Deep Learning
JSON = JavaScript Object Notation
KL = Karhunen-Loéve
ML = Machine Learning
MLP = Multilayer Perceptron
MNIST = Modified National Institute of Standards and Technology
NN = Neural Network
PCA = Principal Component Analysis
RAM = Random-Access Memory
ReLU = Rectified Linear Unit
UV = Ultra Violet

Introducere

Motivație

Polenul reprezintă particule mici produse de majoritatea speciilor vegetale. Acestea conțin celule reproducătoare bărbătești și au un rol foarte important în procesul de reproducere al plantelor.

Studiile privind polenul sunt utilizate pe scară largă în cercetarea alergiilor și combustibililor fosili, știința criminalistică și epidemiologică, industria alimentară, farmaceutică, cosmetică și multe altele [1].

Principalul studiu care ne poate afecta cel mai mult viața este acela în industria alimentară, în special agricultura, deoarece pentru a hrăni populația în creștere a lumii avem nevoie de tot mai multe alimente diverse, echilibrate și de bună calitate pentru a asigura progresul și bunăstarea populației. 35% din culturile alimentare din lume depind direct de procesul de polenizare al plantelor pentru a produce alimente [2]. Potrivit estimărilor unui studiu internațional realizat în 2016 de Platforma interguvernamentală de știință-politică privind biodiversitatea și serviciile ecosistemice, producția anuală globală de alimente care depinde direct de polenizare a valorat între 235 și 577 miliarde de dolari americani [3].

Obiective

Având în vedere importanța polenului în diversele domenii de studiu și impactul asupra agriculturii și economiei la nivel global, obiectivul principal al acestei lucrări este dezvoltarea unui clasificator de particule de polen utilizând algoritmi de clasificare clasici, respectiv algoritmi de învățare profundă. Acest sistem aducând contribuții pentru înțelegerea diverselor specii de polen și întelegerea eficientizării procesului de polenizare.

Structura lucrării

- *Capitolul 1: Noțiuni teoretice/Metode propuse.*

  Prezintă din punct de vedere teoretic algoritmi de clasificare clasici studiați și aplicați pentru clasificarea particulelor de polen, algoritmi de învățare profundă și arhitectura rețelei neurale de la care am pornit și verificarea acestei rețele pe o bază de date cunoscută precum MNIST (Modified National Institute of Standards and Technology database). De asemenea acest capitol conține și o descriere succintă a tehnologiilor utilizate pentru realizarea proiectului.

- *Capitolul 2: Clasificarea automată a polenului. Baza de date.*

  Descrie problema clasificării automate a polenului și cum este obținută baza de date. Descrierea bazei de date neprelucrată și procesul de prelucrare al acesteia.

- *Capitolul 3: Clasificarea automată a polenului. Metode, arhitecturi și rezultate.*

  Conține descrierea detaliilor de implementare/modul de desfășurare al experimentelor(definirea modelului, antrenarea, testarea, optimizarea hiperparametrilor/adaptarea arhitecturii) și prezentarea rezultatelor obținute.

- *Concluzii.*

  Prezentarea concluziilor generale și a contribuțiilor personale. Detalii asupra dezvoltării viitoare a proiectului.

Capitolul 1
Noțiuni teoretice/Metode propuse

În acest capitol este prezentată partea teoretică a algoritmilor de clasificare clasici (în subcapitolul 1.1) și a algoritmilor de învățare profundă (în subcapitolul 1.2).

1.1 Algoritmi de clasificare clasici (Machine learning)

Pentru a avea un model de referință în evaluarea performanțelor clasificatorilor neurali de învățare profundă, acest capitol prezintă succint clasificatorii clasici aleși a căror funcționare nu implică aplicarea teoriei rețelelor neurale (Algoritmul Bayes naiv, Algoritmul celor mai apropiați vecini, Algoritmul proces gaussian) și cei mai puțin complecși clasificatori clasici neurali (perceptron, perceptron multi nivel).

1.1.1 Algoritmul Bayes naiv

Algoritmul Bayes naiv este un model supervizat care se bazează pe aplicarea teoremei lui Bayes (1.1), care ne permite să calculăm probabilități condiționate. Ne permite să folosim cunoștințe/probabilități apriori (rezultate din setul de antrenare) pentru a calcula probabilitatea de apariție a unui eveniment corelat.

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)} \tag{1.1}$$

Acest algoritm se bazează pe presupunerea "naivă" că observațiile/caracteristicile fiecărei clase sunt independente și respectă exact o anumită distribuție spațială.

Diferitele clasificatoare Bayes naiv diferă în principal prin ipotezele care se fac cu privire la distribuția spațială. Pentru experimentele realizate pe acest algoritm am ales o distribuție gaussiană (valorile continue asociate cu fiecare caracteristică sunt presupuse a fi distribuite în conformitate cu o distribuție normală).

Cel mai întâlnit caz al clasificatorului Bayes corespunde situației în care avem doar două clase de clasificat (ipoteză binară), deoarece multe situații de clasificare crează în mod natural două clase (corect/greșit, obiect prezent în imagine/obiect absent în imagine). Deși această metodă poate fi ușor generalizată la mai mult de două clase [4].

1.1.2 Algoritmul celor mai apropiați vecini

Algoritmul celor mai apropiați k vecini reprezintă un algoritm de învățare supervizat, care clasifică un eșantion (vector de intrare) la o clasă prin a verifica cea mai frecventă clasă dintre clasele celor k exemple de antrenament, cele mai apropiate de exemplul de testare. Parametrul k este o constantă specificată de utilizator și reprezintă numărul de eșantioane din setul de antrenament care să fie verificate în jurul eșantionului de testare (atribuie celor mai apropiați k vecini o pondere de apartenență la aceeași clasă egală cu 1/k și la restul o pondere 0).

Acest parametru se alege în funcție de setul de date și este recomandat să fie un număr impar pentru a elimina incertitudinile în cazul voturilor egale [5].

Etapa de antrenare pentru acest algoritm constă doar în memorarea eșantioanelor de antrenament și etichetarea corespunzătoare a acestora. Astfel acest algoritm prezintă o antrenare rapidă, dar o testare care necesită timp, deoarece partea computațională are loc în timpul testării.

1.1.3 Algoritmul proces gaussian

Algoritmul proces gaussian este un algoritm supervizat care calculează o predicție probabilistică (gaussiană), predicțiile de testare iau forma probabilităților clasei. Dezavantajul principal al acestei metode care poate influența aplicarea acesteia pentru setul de date de polen este acela că folosește toate informațiile despre eșantioane/caracteristici pentru a efectua predicția și își pierde eficiența în spații cu dimensiuni ridicate [6].

1.1.4 Perceptron

Perceptronul reprezintă cel mai simplu clasificator clasic neural supervizat. Acesta dispune de un lot de antrenare etichetat unde etichetele asociate vectorilor de intrare reprezintă codificarea clasei căreia îi aparțin (ieșirea ideală a rețelei). În procesul de antrenare, pentru fiecare vector aplicat la intrarea rețelei se calculează ieșirea și este comparată cu eticheta pentru a calcula eroarea și corecțiile care trebuie aplicate ponderilor sinaptice astfel încât să se minimizeze această eroare.

Arhitectura unei rețele perceptron (Figura 1.1) este formată din stratul de intrare care nu prezintă parte computațională, acest strat fiind compus din valorile vectorilor de intrare și un strat de ieșire. Stratul de ieșire este format din părți mai mici numite neuroni. Numărul de neuroni din stratul de ieșire fiind determinat de numărul de clase.

Legăturile dintre aceste două straturi succesive sunt realizate prin intermediul unor ponderi, care sunt adaptate în urma procesului de antrenare.

Pentru o alegere binară rețeaua neurală perceptron prezintă un singur neuron pe stratul de ieșire.

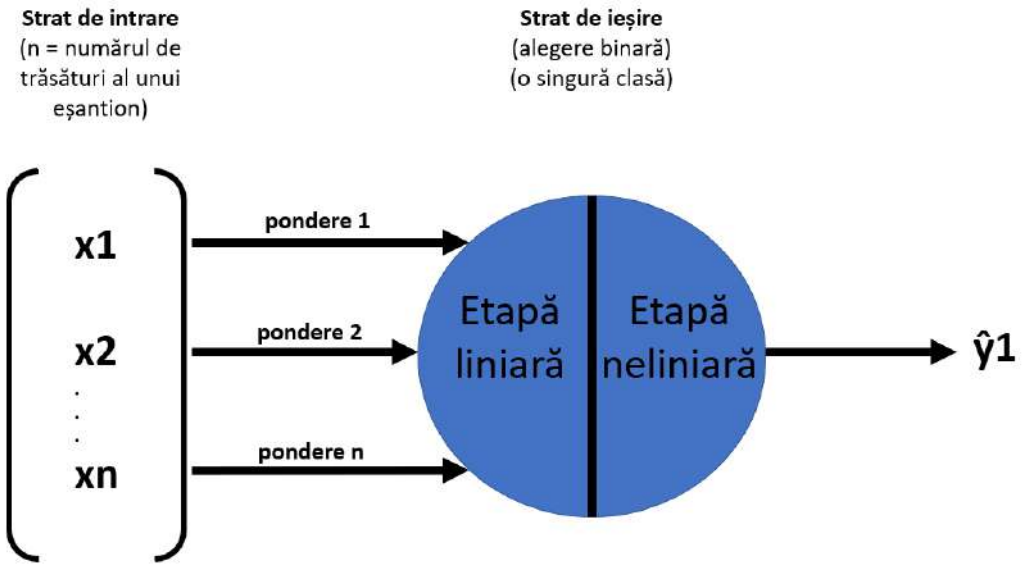


Figura 1.1: Rețea neurală de tip perceptron

Fiecare neuron este compus dintr-o etapă liniară a cărui ieșire este calculată astfel:

$$Z = W^\top X + b \tag{1.2}$$

În ecuația 1.2, X reprezintă vectorul de intrare în rețea, iar W și b reprezintă ponderile de legătura între cele două straturi. În cazul din figura 1.1 W reprezintă un vector de n ponderi, acesta fiind transpus pentru a putea fi înmulțit cu vectorul coloană X, iar b reprezintă un scalar. Ponderile w1, w2, ... wn sunt inițializate aleator după o distribuție normală, iar ponderea b este inițializată cu 0.

După partea liniară a neuronului urmează o etapă neliniară (ecuația 1.3) în care aplicăm o funcție de activare pe ieșirea etapei liniare pentru a obține la ieșire rețelei o probabilitate.

$$\hat{y} = f(Z) \tag{1.3}$$

Cele mai întâlnite funcții de activare sunt regăsite în Figura 1.2.

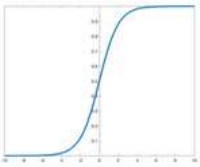
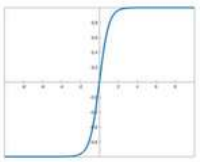
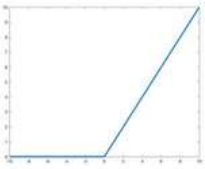
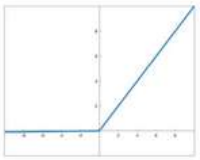
Activation function	Equation	Graph
Sigmoid	$S(x) = \frac{1}{1+e^{-x}}$	
Tanh	$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	
ReLU	$RELU(x) = \begin{cases} 0 \text{ if } x < 0 \\ x \text{ if } x >= 0 \end{cases}$	
Leaky ReLU	$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$	

Figura 1.2: Exemple de funcții de activare
[7]

Antrenarea rețelei perceptron constă într-o etapă de calculare a erorii (funcția cost) între etichetele de antrenare și ieșirea rețelei. Și o etapă de adaptare a ponderilor pentru a minimiza funcția cost. Ponderile sunt adaptate cu o rată de antrenare, hiperparametru ales de utilizator.

Funcția loss reprezintă eroarea pentru un singur exemplu de antrenare.

Funcția cost reprezintă media funcțiilor loss pentru întregul set de învățare.

După ce terminăm etapa de antrenare a rețelei, salvăm ponderile obținute și trecem la etapa de testare, din care obținem o statistică a acurateții modelului antrenat.

1.1.5 Perceptron multi strat

Perceptronul multi nivel aduce un grad mai mare de complexitate rețelei perceptron simplă prin adăugarea straturilor ascunse între stratul de intrare și stratul de ieșire. Numărul de straturi ascunse și numărul de neuroni din care acestea sunt formate având valori arbitrare (reprezentând doi hiperparametrii foarte importanți).

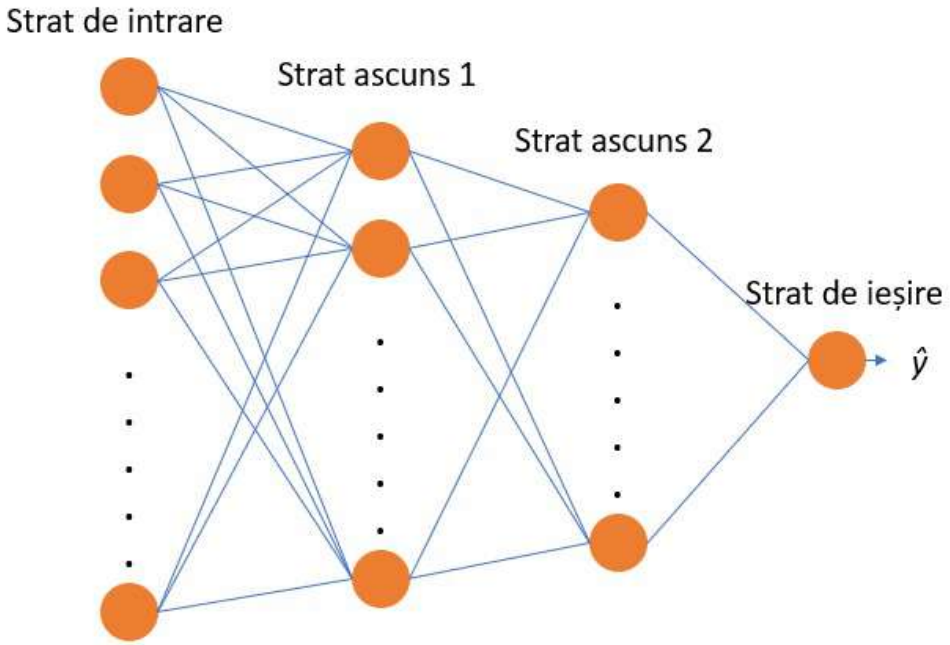


Figura 1.3: Exemplu arhitectură MLP

În Figura 1.3 este reprezentată o rețea neurală de tip MLP (Multilayer Perceptron). Structura internă a neuronilor și procesele de antrenare și validare sunt asemenătoare cu cele ale unei arhitecturi perceptron simple, diferența constă în numărul de ponderi sinaptice mult mai ridicat.

1.2 Algoritmi de învățare profundă

Pentru a încerca depășirea performanțelor algoritmilor de ML (Machine Learning) clasici am apelat și la implementarea unor algoritmi de învățare profundă ce pot fi utilizați pentru sarcina curentă precum CNN (Convolutional Neural Network).

1.2.1 Rețele neurale convoluționale

Rețelele neurale convoluționale reprezintă o categorie de rețele neurale adânci care utilizează straturi de tip convoluțional pentru filtrarea datelor de intrare. Aceastea sunt folosite în practică atunci când avem imagini ca date de intrare, deoarece acestea funcționează precum un detector de contur și extrag informația asupra distribuției spațiale a imagini.

O rețea convoluțională prezintă 3 tipuri de straturi:

- **Straturi convoluționale**

  Straturile convoluționale au rolul de a prelucra informația de intrare cu ajutorul operației de convoluție folosind diverse filtre de detecție.

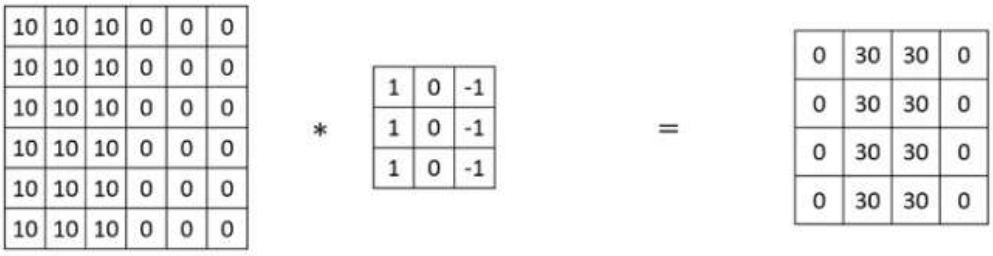


Figura 1.4: Exemplu detector de contur folosind operația de convoluție [8]

În Figura 1.4 este reprezentat rezultatul aplicării unui filtru convoluțional de tip detector de contur vertical 3x3 pe o imagine de intrare de dimensiuni 6x6.

Dimensiunea caracteristicii rezultate poate fi calculată cu formula 1.4.

$$out = \frac{in + 2p - f}{s} + 1 \tag{1.4}$$

unde:

  - in = dimensiunea caracteristicii de intrare
  - p = padding (dimnesiunea bordării cu 0 a intrării)
  - f = dimensiunea filtrului convoluțional
  - s = stride (pasul de deplasare al filtrului pe intrare)

- **Straturi de extragere (pooling)**

  Acestea au scopul principal de a reduce dimensiunea hărților de caracteristici. Acest strat nu prezintă parametri de optimizat. În practică un strat convoluțional este urmat de un strat de extragere. Cele mai întâlnite filtre pentru acest strat sunt filtre de tip extragere maxim și extragere medie de dimensiune 2x2 cu pas 1 și bordare 0.

- **Straturi dense**

  Partea straturilor dense este întâlnită la finalul rețelelor convoluționale pentru calcularea probabilităților de apariție la clase. Acestea au o structură similară cu cele ale unui MLP.

1.2.2 Arhitectura LeNet-5

LeNet-5 este una dintre primele arhitecturi de rețele neurale convoluționale și a ajutat foarte mult la dezvoltarea algoritmilor de învățare profundă. Aceasta a fost publicată în 1989 de Yann LeCun [9].

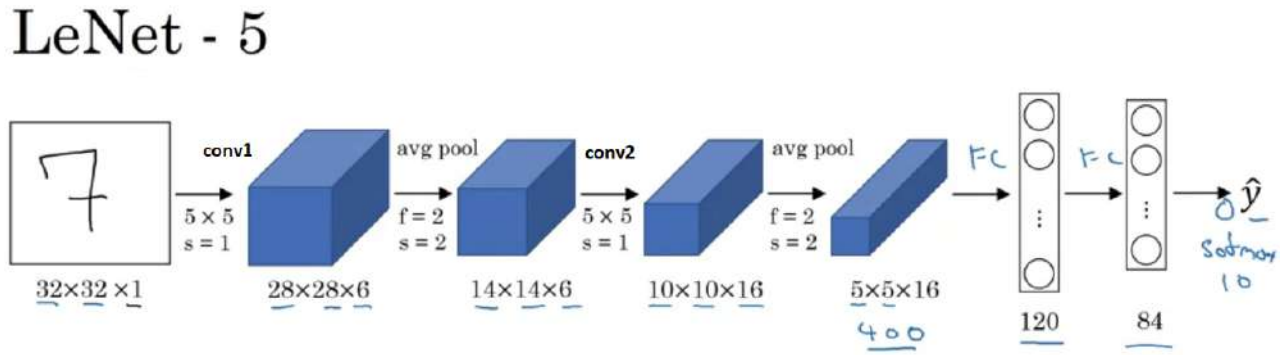


Figura 1.5: Arhitectura LeNet-5
[8]

Arhitectura ilustrată în Figura 1.5 prezintă 5 straturi dintre care primele doua sunt formate dintr-un strat convoluțional și un strat de extragere (pool), urmate de 3 straturi dense. Arhitectura a fost construită pentru clasificarea imaginilor în tonuri de gri de dimnesiune 32x32 care conțin numere de la 0 la 9.

1.2.3 LeNet-5 aplicat pentru clasificare cifrelor MNIST

Pentru verificarea implementării corecte a arhitecturii LeNet-5 am efectuat primele experimente pe o bază de date cunoscută pentru a verifica dacă performanțele obținute de către mine sunt similare cu rezultatele altor lucrări asemănătoare.

Setul de date MNIST este format din imagini de dimensiuni 28x28 care conțin numere de la 0 la 9 scrise de mână (Figura 1.6). Această bază de date conține un set foarte mare de eșantioane etichetate egal cu 70.000 dintre care 60.000 sunt folosite pentru antrenarea rețelei și 10.000 pentru testarea performanțelor.

Figura 1.6: Exemple eșantioane din baza de date MNIST
[10]

Capitolul 2
Clasificarea automată a polenului. Baza de date

În acest capitol este descrisă sarcina de clasificare automată a polenului și complexitatea acesteia (în subcapitolul 2.1), apoi sunt descrise caracteristicile bazei de date SAU-SRB (în subcapitolul 2.2) și în final, în subcapitolul 2.3 am prezentat metodele de preprocesare a datelor.

2.1 Clasificarea automată a polenului

Clasificarea automată a particulelor de polen provenite de la diferite specii este o sarcină dificilă care este de obicei abordată manual de către operatori umani folosind microscopul. Multe industrii din cele enumerate în capitolul de introducere, inclusiv cele medicale și farmaceutice, se bazează pe acuratețea acestui proces manual de clasificare. Această acuratețe aflându-se în jurul valorii de 67% [11].

De asemenea, și sarcina de obținere a particulelor de polen este una destul de dificilă. Aceasta fiind abordată diferit de-a lungul istoriei cu diferite instumente de măsură. Primele raportări ale unui asemenea instrument datează de la inceputul secolului al 19-lea, precum cercetătorul Blackley care a experimentat cu un instrument compus din lamele de sticlă acoperite cu un amestec de glicerină pentru o perioadă de 24 de ore, care erau ulterior analizate manual folosind un microscop [12]. O îmbunatățire majoră a studiului polenului a fost dezoltarea capcanei Hirst care încă este folosită pentru monitorizarea polenului. Acest sistem constă într-o pompă de aer și o rolă de bandă acoperită în vaselină care se mișcă la o viteză constantă [13].

Având în vedere complexitatea sarcinilor de obținere și clasificare manuală, studiul polenului se îndreaptă spre sisteme automate.

Un astfel de sistem automat capabil să numere particule de polen este instrumentul Rapid-E dezvoltat de Plair. Acesta utilizează mai multe lasere pentru a determina dimensiunea și forma particulelor de polen.

Pentru o singură particulă de polen măsurată de aparatul Rapid-E, acesta procesează următoarele 3 caracteristici:

O **imagine de împrăștiere** multi-unghi 24 de unghiuri [-45, 45], de dimensiune variabilă deoarece dimensiunea imaginii depinde de dimensiunea particulei de aerosol. Aceasta este generată cu ajutorul unei surse laser, lentile și senzori fotodectori. Figura 2.1.

Un **spectru de fluorescență** cu 32 de canale/lungimi de undă înregistrate la 8 momente de timp, realizat de o sursă laser UV (Ultra Violet) și un fotodetector cu 32 de canale. Figura 2.1.

Un **timp de răspuns fluorescent** care corespunde cu diferența de timp dintre momentul în care un puls de laser UV este trimis și momentul de răspuns fluorescent al particulei. Acesta este măsurat pentru 4 lungimi de undă diferite: 350-400nm, 420-460nm, 511-572nm, 672-800nm înregistrate într-un interval de timp constant. Această caracteristică este măsurată de fotodetectori care operează pe 4 canale. [14]. Figura 2.1

În această lucrare am folosit setul de date numit SAU-SRB provenit de la un instrument Rapid-E aflat în Novi Sad - Serbia [15].

2.2 Descrierea bazei de date SAU-SRB

Setul de date SAU-SRB este format din 14 clase/specii de polen diferite. Fiecare clasă având un număr diferit de eșantioane, conform Tabelului 2.1.

Număr	Specii de polen	Numărul de eșantioane
1	Acer	4399
2	Alnus	17471
3	Artemisia	25486
4	Betula	27347
5	Cedrus	5761
6	Corylus	26035
7	Cynodon	2728
8	Dactylis	5861
9	Fraxinus	10230
10	Picea	9702
11	Populus	16945
12	Quercus	4194
13	Salix	15755
14	Taxus	10373

Tabela 2.1: Baza de date SAU-SRB neprelucrată

În total, setul de date SAU-SRB inițial cuprinde un număr de **182.287** eșantioane. Pentru fiecare eșantion dispunem de următoarele caracteristici:

- o *imagine de împrăștiere* (dimensiune variabilă X 24);
- un *spectru de fluorescență* (32 X 8);
- un *timp de răspuns fluorescent* (64 X 4).

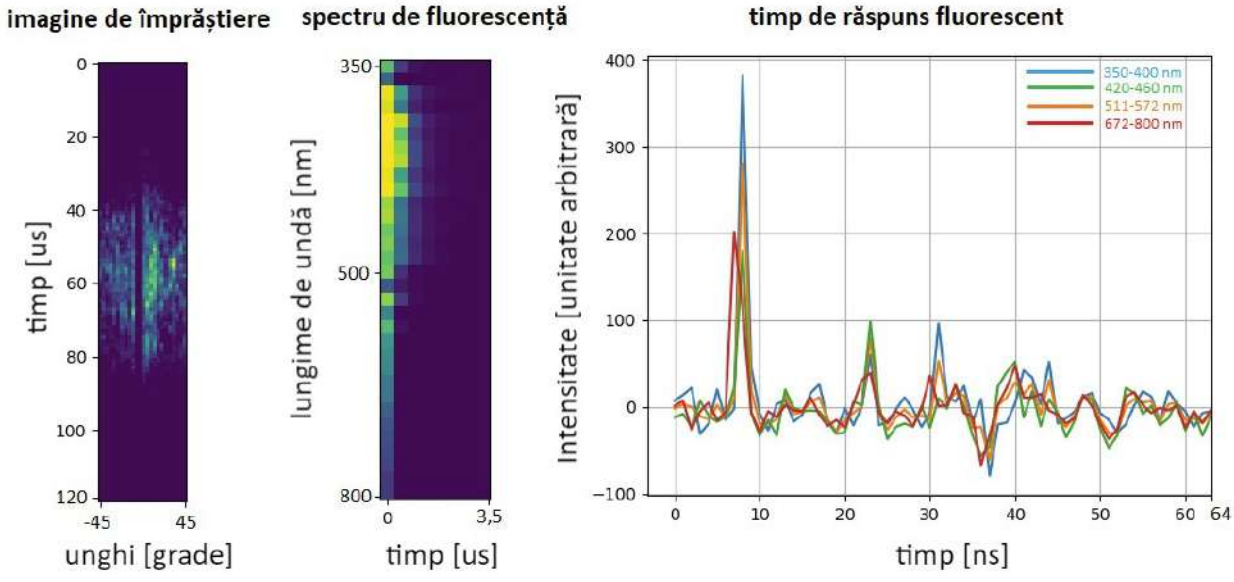


Figura 2.1: Vizualizarea celor 3 caracteristici din setul de date SAU-SRB

2.3 Preprocesarea datelor

Pentru a înțelege cum pot filtra și prelucra setul de date, am început prin a vizualiza diferitele caracteristici.

- **Vizualizarea imaginilor de împrăștiere.** (Figura 2.2)

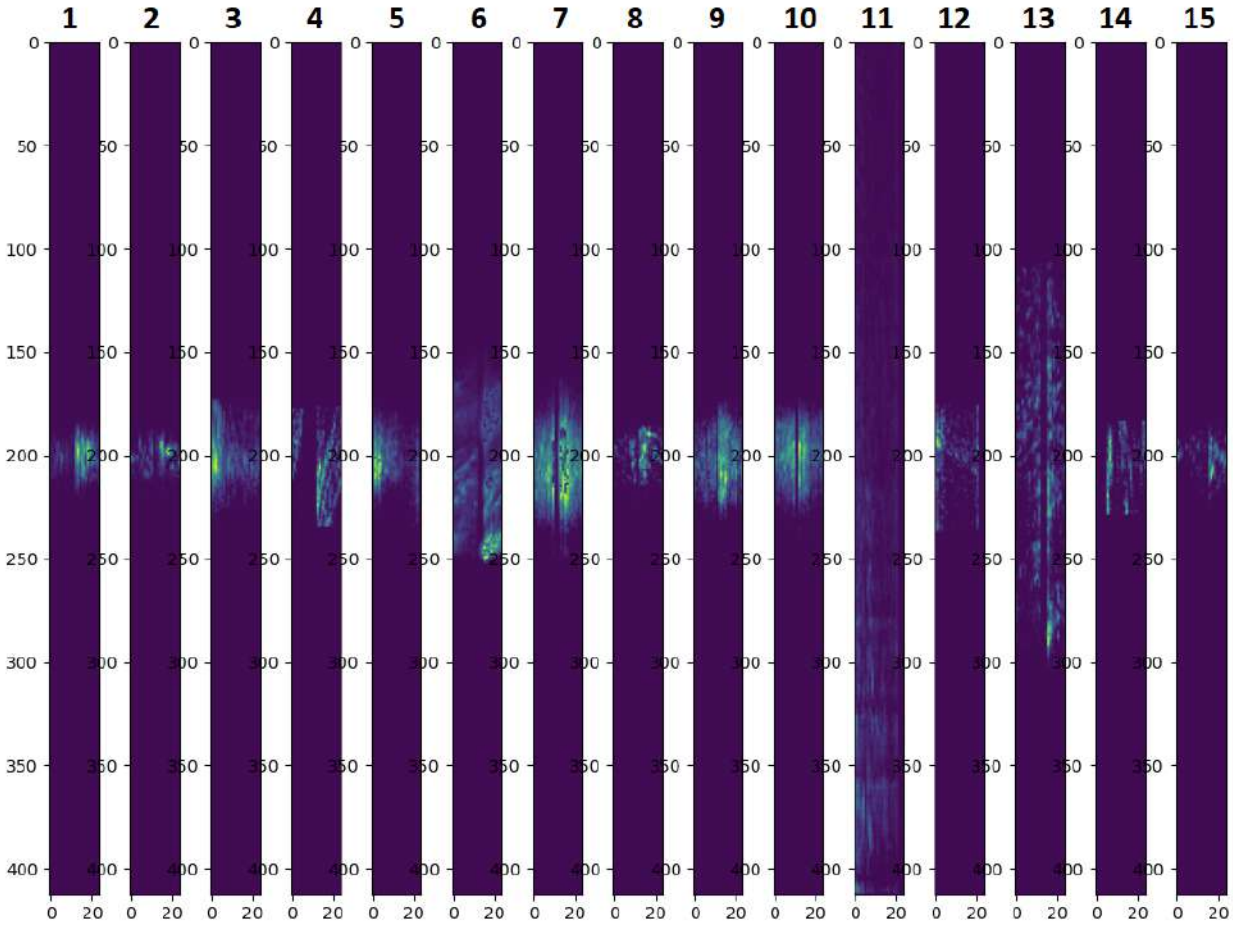


Figura 2.2: Vizualizarea a 15 imagini de împrăștiere din clasa Cynodon

Deoarece prima dimensiune a imaginilor variază în funcție de dimensiunea particulei de aerosol, această dimensiune ajungând pentru unele eșantioane până la valoarea maximă de 25392, am ales pentru vizualizare o valoare apropiată de media dimensiunilor eșantioanelor.

Dimensiunea imaginilor a fost selectată 420 x 24 (dimensiunea particulei X numărul de unghiuri). Pentru imaginile a căror dimensiune inițială a fost mai mică decât 420 a fost nevoie de o bordare cu 0 pentru vizualizarea lor.

Din această vizualizare (Figura 2.2) putem observa că în datele prelucrate de aparatul de măsură apar imagini zgomotoase precum eșantioanele 4, 5, 11, 12, 13, 14 și respectiv 15, care trebuie filtrate din setul de date deoarece ne așteptăm ca particulele de polen să aibă o imagine de împrăștiere simetrică.

- **Vizualizarea spectrelor de fluorescență.** (Figura 2.3)

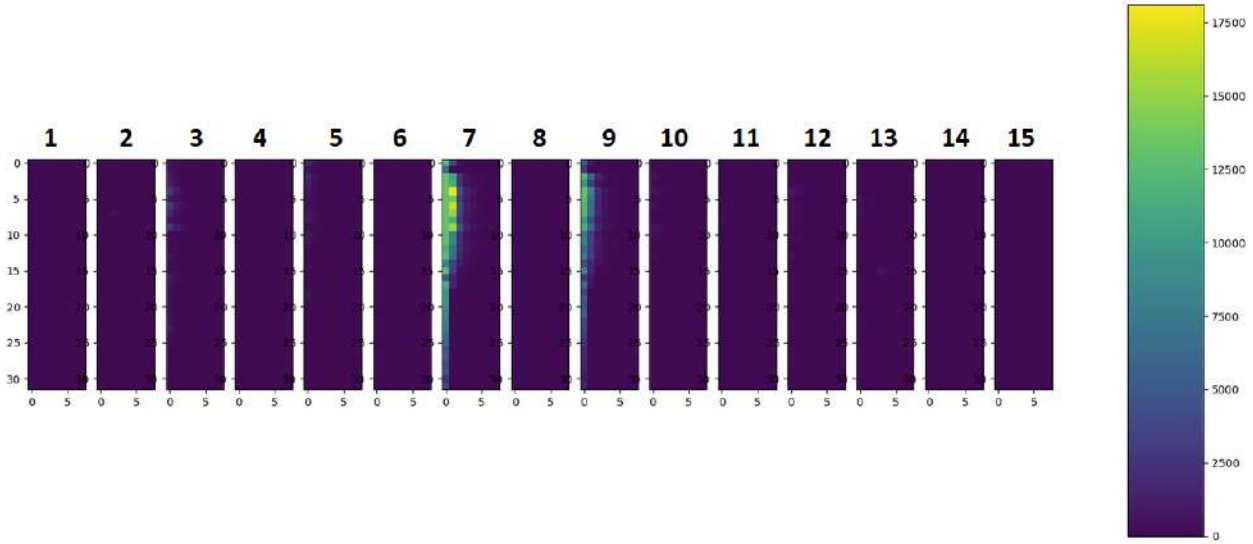


Figura 2.3: Vizualizarea a 15 spectre de fluorescență din clasa Cynodon

Dimensiunea spectrului de imagine este 32X8 (32 de lungimi de undă înregistrate la 8 momente de timp)

Din vizualizarea spectrelor de fluorescență (Figura 2.3) observăm că majoritatea eșantioanelor au un răspuns fluorescent scăzut, iar acestea corespund imaginilor de împrăștiere zgomotoase.

2.3.1 Filtrarea și prelucrarea setului de date

După vizualizarea diferitelor caracteristici s-au putut identifica unele eșantioane eronate în setul de date, eșnatioane care provin cel mai probabil din măsurarea automată a unor particule de aerosol diferite față de cele de polen.

Pentru a înlătura aceste eșantioane eronate am selectat manual câte un prag de separare pentru cele două caracteristici vizualizate;

- Pentru **imaginilie de împrăștiere** am ales dimensiunea maximă a unei particule de aerosol 120. Eșantioanele care conțin imagini de împrăștiere de dimensiuni mai mari de 120 sunt considerate eșantioane eronate și sunt șterse din baza de date, iar particulele cu o dimensiune mai mică decât pragul selectat vor fi bordate cu zero pentru a avea toate imaginile de dimensiune constantă și anume 120X24 (dimensiunea particulei X numărul de unghiuri).

  În urma aplicării acestui prag au fost eliminate **15.016** de eșantioane.

- Pentru **spectrele de fluorescență** am observat că valorile variază în intervalul 0 - 17.500, iar eșantioanele eronate corespundeau unor spectre de fluorescența cu valori mici. De aceea am ales criteriul de separare: orice eșantion al carui spectru de fluorescență nu conține nici o valoare mai mare decât 1.000 este considerat eșantion eronat și va fi șters din baza de date.

  În urma aplicării acestui prag au fost eliminate **80.758** de eșantioane.

Numărul inițial de eșantioane = **182.287**
Numărul de eșantionare după filtrare = 182.287 - 15.016 - 80.758 = **86.513**

Deoarece numărul de eșantioane al claselor este foarte diferit - Tabelul 2.2 (setul de date nu este echilibrat), cea mai ușoară soluție de a echilibra puțin setul de date a fost normalizarea.

Am normalizat setul de date pentru fiecare caracteristică prin a împarții la valoarea maximă din respectiva caracteristică. Această prelucrare îmi va scala datele în intervalul [0, 1] pentru valori strict pozitive, respectiv în intervalul [-1, 1] pentru valori pozitive și negative.

Normalizarea oferă importanță/ponderi egale fiecărei variabile, astfel încât performanța modelului să nu fie mai puternic influențată de o anumită caracteristică.

Număr	**Specii de polen**	**Numărul inițial de eșantioane**	**Numărul de eșantioane după filtrare**	**Ponderea claselor raportată la dimensiunea întregului set de date**
1	Acer	4399	1335	1,54%
2	Alnus	17471	3874	4,47%
3	Artemisia	25486	15209	17,58%
4	Betula	27347	12544	14,49%
5	Cedrus	5761	1236	1,42%
6	Corylus	26035	15760	18,21%
7	Cynodon	2728	1302	1,5%
8	Dactylis	5861	1952	2,25%
9	Fraxinus	10230	5599	6,47%
10	Picea	9702	2456	2,83%
11	Populus	16945	7070	8,17%
12	Quercus	4194	1721	1,98%
13	Salix	15755	10086	11,65%
14	Taxus	10373	6369	7,36%

Tabela 2.2: Statistica bazei de date SAU-SRB după aplicarea filtrării

Baza de date primită (SAU-SRB) este salvată în 14 fișiere de tip JSON (JavaScript Object Notation), un fișier pentru fiecare clasă. În fiecare fișier aflându-se o listă de dicționare. Un dicționar reprezentând un eșantion. Dicționarul fiind la rândul lui format din cele 3 caracteristici prezentate (imaginea de împrașiere, spectrul de fluorescență și timpul de răspuns fluorescent) plus data la care a fost procesat eșantionul.

Key	Type	Size	Value
Lifetime	list	256	[-19, -13, -9, -3, 2, 8, 13, -4, -23, -39, ...]
Scattering	dict	1	{'Image':[0, 0, 0, 0, 0, ...]}
Spectrometer	list	256	[45, 91, 98, 0, 30, 22, 0, 38, 7, 51, ...]
Timestamp	str	23	2018-04-17 13:31:00.667

Figura 2.4: Formatul inițial al unui eșantion din baza de date SAU-SRB

După cum se poate vedea în Figura 2.4 fiecare caracteristică este vectorizată și salvată într-o listă, mai puțin imaginea de împrastiere, aceasta fiind salvată într-un dicționar cu o singură cheie atribuită listei de valori vectorizate ale imaginii de dimensiune variabilă.

Baza de date prelucrată a fost salvată în format NPY pentru un acces mai rapid și mai simplu al datelor în scripturile de Python realizate.

Timpul de încărcare al bazei de date inițiale de tip JSON = 47,56 secunde

Timpul de încărcare al bazei de date prelucrată de tip NPY = 0,78 secunde

Index	Type	Size
0	Array of int32	(3392, 1335)
1	Array of int32	(3392, 3874)
2	Array of int32	(3392, 15209)
3	Array of int32	(3392, 12544)
4	Array of int32	(3392, 1236)
5	Array of int32	(3392, 15760)
6	Array of int32	(3392, 1302)
7	Array of int32	(3392, 1952)
8	Array of int32	(3392, 5599)
9	Array of int32	(3392, 2456)
10	Array of int32	(3392, 7070)
11	Array of int32	(3392, 1721)
12	Array of int32	(3392, 10086)
13	Array of int32	(3392, 6369)

Figura 2.5: Formatul bazei de date prelucrate

Am salvat 14 fișiere NPY, unul pentru fiecare clasă. Un fișier NPY conține o matrice de dimensiune **(3.392 X Numărul de eșantioane al clasei respective)** (Figura 2.5).

Iar după încărcarea acestora în scripturile de antrenare datele sunt concatenate formând astfel o matrice de dimensiune **3.392 X 86.513 (numărul de trăsături X numărul de eșantioane/observații)**. Acest proces este reprezentat grafic în figura 2.6.

Dimensiunea 3.392 a fost obținută prin concatenarea celor trei caracteristici vectorizate:

(120X24) + (32X8) + (64X4) = 3.392

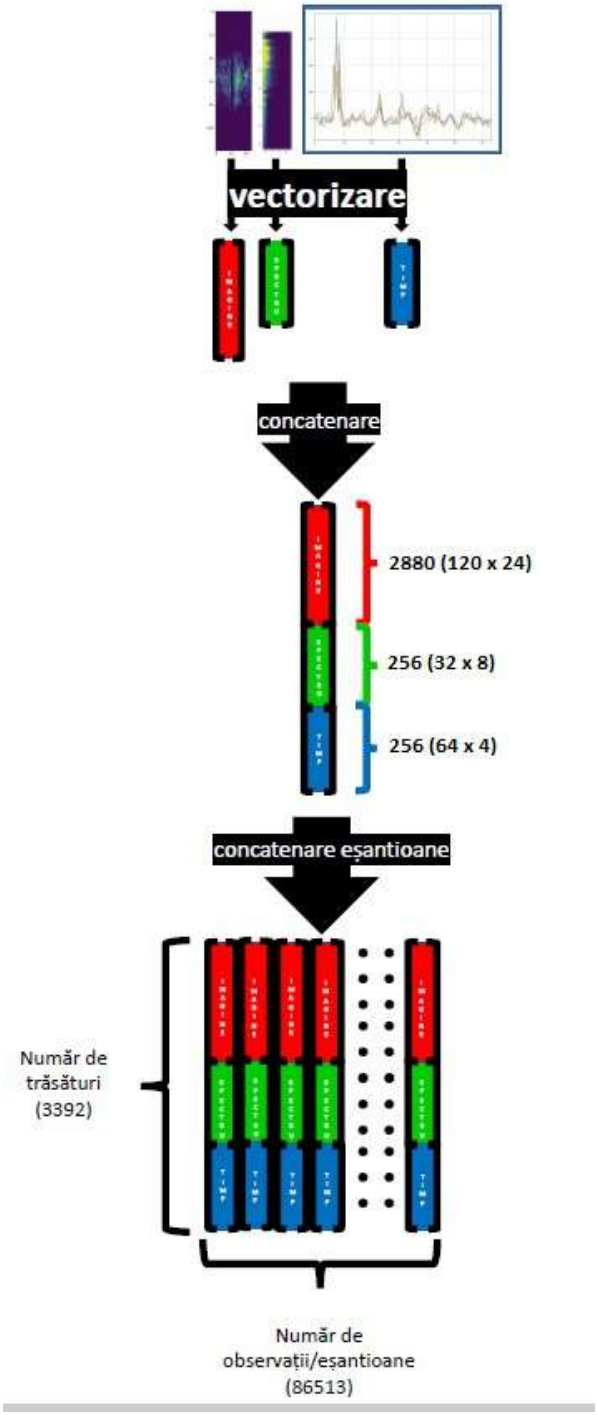


Figura 2.6: Reprezentarea grafică a procesului de salvare a bazei de date prelucrată

Pentru a mă asigura că în urma filtrării există diferențe între cele 14 clase am reprezentat grafic distribuția valorilor fiecărei caracteristici pentru fiecare clasă și pentru toate clasele per ansamblu.

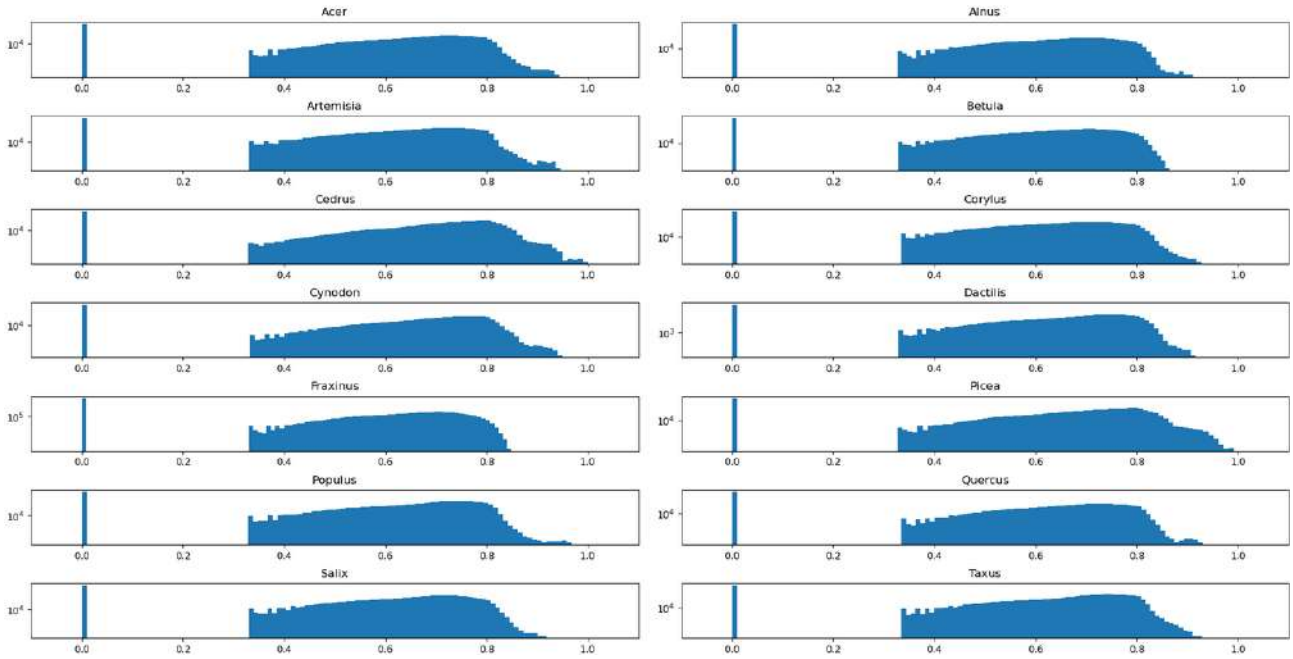


Figura 2.7: Histogramele imaginilor de împrăștiere pentru fiecare clasă

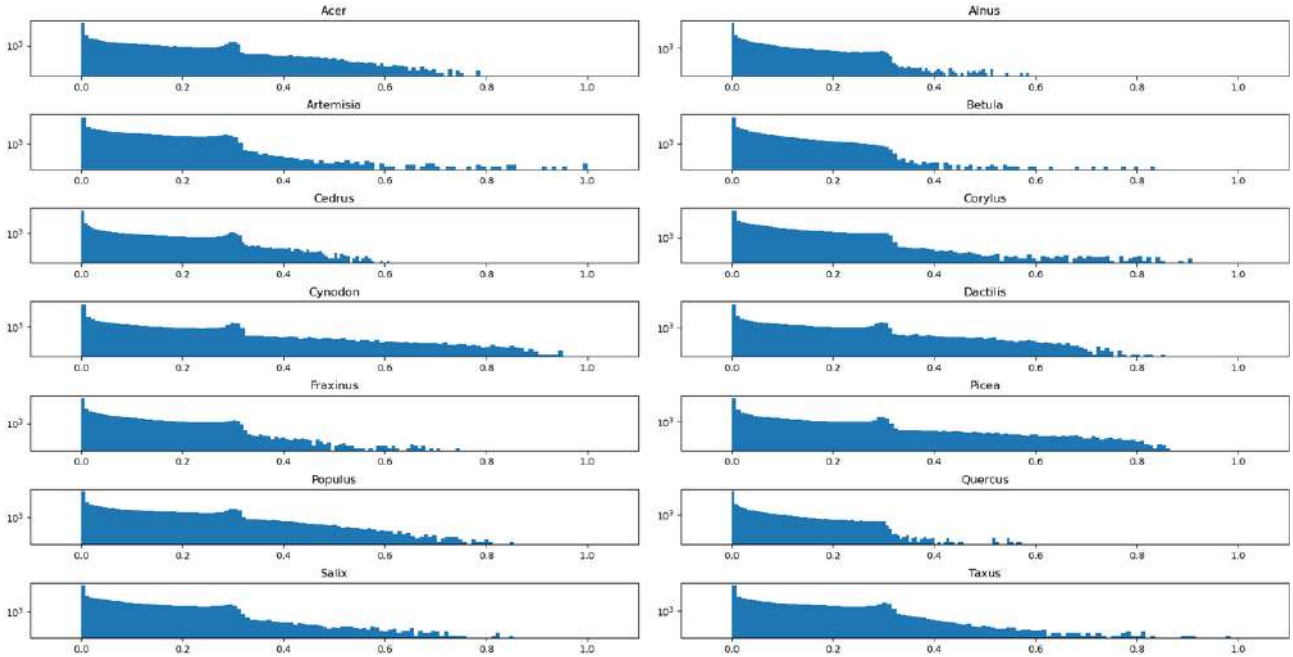


Figura 2.8: Histogramele spectrelor de fluorescență pentru fiecare clasă

Se observă din Figura 2.7 că distribuția valorilor este ușor diferită pentru fiecare clasă. De asemenea, se poate observa că pragul ales pentru filtrarea imaginilor nu a scos eșantioane de polen deoarece în fiecare clasă are loc bordarea cu zero a imaginilor.

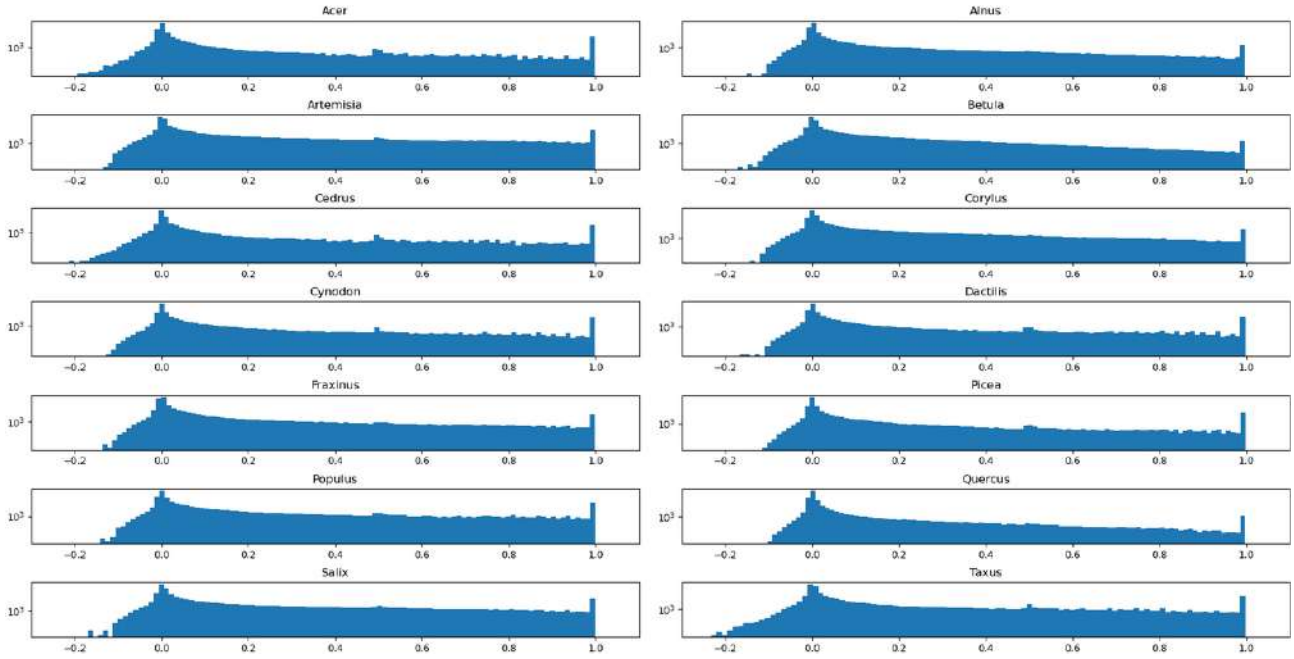


Figura 2.9: Histogramele timpilor de răspuns fluorescent pentru fiecare clasă

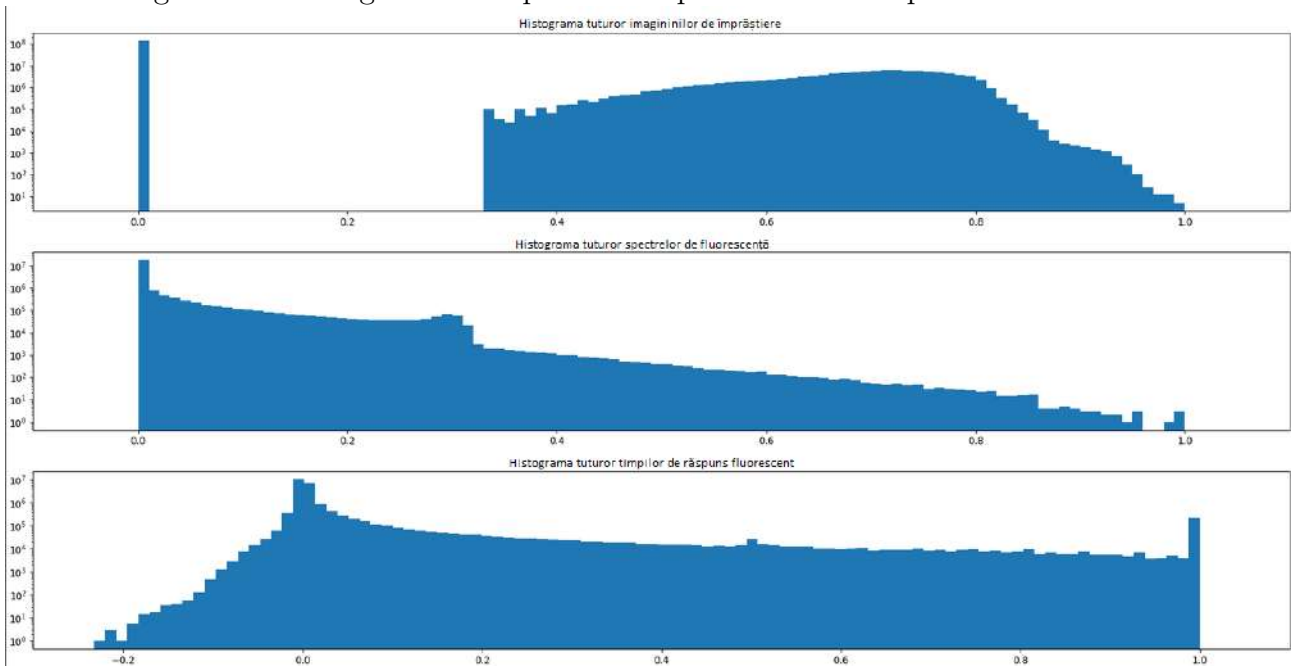


Figura 2.10: Histogramele celor 3 caracteristici pentru tot setul de date

Se observă din figurile 2.7, 2.8 și 2.9 că fiecare clasă are o distribuție a valorilor asemenătoare cu cele 3 distribuții din tot setul de date din Figura 2.10. Totuși, acestea variază ușor de la o clasă la alta, prin urmare am putea dezvolta un clasificator automat care să se folosească de aceste mici diferențe.

După familiarizarea cu setul de date SAU-SRB și preprocesarea cu succes a acestuia, următorul pas de prelucrare a datelor de intrare pentru algoritmii de clasificare clasici, respectiv de învățare profundă este organizarea datelor în seturi de antrenare, testare și validare.

Cele 14 matrice NPY au fost prima dată etichetate prin crearea unei liste de etichete de dimensiune 86513 (numărul de eșantioane al bazei de date), apoi concatenate într-o matrice de dimensiune 3392 X 86513 (vezi Figura 2.5). Apoi, eșantioanele au fost amestecate pentru a mă asigura că fiecare iterație în procesul de antrenare crează o modificare independentă, astfel încât performanța modelului să nu fie influențată de eșantioanele consecutive de la o singură clasă.

- Pentru **algoritmii de clasificare clasici** am împarțit setul de date în 80% set de antrenare, 10% set de testare și 10% set de validare.
- Pentru **algoritmii de învățare profundă** (CNN) am împarțit setul de date în 80% set de antrenare și 20% set de testare.

2.3.2 Analiza componentelor principale

Deoarece datele de intrare au o dimensiune semnificativă constând din informațiile concatenate de la 3 caracteristici, iar algoritmii clasici de ML pot calcula mai bine și mai rapid pe date de dimensiune mai mică, am decis să introduc în experimentele efectuate pentru această lucrare și aplicarea algoritmului de analiză a componentelor principale. Deoarece am ales și algoritmi de ML clasici mai puțin complecși aceștia pot avea performanțe slabe pe un set de date de dimensionalitate crescută precum setul SAU-SRB.

Analiza componentelor principale (PCA) reprezintă o metodă prin care se poate reduce dimensionalitatea datelor de intrare păstrând în același timp cea mai mare cantitate de informație.

Metoda PCA se bazează pe transformata Karhunen-Loéve (KL). KL este o transformare liniară cu proprietăți de conservare a energiei care este esențială pentru a reduce dimensionalitatea datelor și a păstra cât mai multă informație [16].

Capitolul 3
Clasificarea automată a polenului. Metode, arhitecturi și rezultate

În acest capitol vom aborda sarcina de clasificare automată a polenului folosind o serie de metode de clasificare clasice (în subcapitolul 3.1) și apoi diverse topologii de rețele convoluționale profunde (CNN, în subcapitolul 3.2). În final, în subcapitolul 3.3 am sumarizat toate rezultatele obținute și am concluzionat că rețelele de tip CNN ce folosesc toate caracteristicile la intrare oferă cele mai bune performanțe.

3.1 Rezultate folosind metode clasice de clasificare

Ideea principală a fost selectarea a 5 algoritmi ML de clasificare clasici și monitorizarea performanțelor acestora pe setul de date SAU-SRB.

Pentru utilizarea algoritmilor în limbajul de programare Python am folosit biblioteca Scikit-learn.

Din algoritmii de clasificare clasici of Scikit-learn am ales să experimentez cu:

- **Algoritmul Bayes naiv**. Am utilizat algoritmul Bayes naiv gaussian cu paramentri impliciți din biblioteca Scikit-learn [17].

- **Algoritmul celor mai apropiați vecini**. Am realizat experimentul algoritmului celor mai apropiați K vecini cu parametrul K = 70 și ponderi uniforme utilizate pentru predicție (toate punctele din fiecare clasă sunt ponderate în mod egal). De asemenea, funcția acestui clasificator și a următorilor 3 din prezentare acceptă și un parametru de intrare care setează numărul de procese care pot lucra în paralel (utilizarea tuturor procesoarelor pentru antrenare) [18].

- **Algoritmul proces gaussian**. Am utilizat clasificatorul proces gaussian cu paramentrii impliciți din biblioteca Scikit-learn [19].

  Problema cu acest algoritm este că necesită estimarea matricelor de covarianță pe întreg spațiul de intrare, motiv pentru care nu este recomandat dacă lucrăm cu date de dimensiune mare precum setul meu de date deoarece antrenarea acestui sistem necesită un spațiu foarte mare de memorie RAM (Random-Access Memory). Pentru a remedia această problemă trebuie să împărțim setul de date în loturi mai mici și să realizăm o antrenare incrementală. Modelul GaussianProcessClassifier din biblioteca Scikit-learn nu prezintă în momentul actual o metodă de utilizare a antrenării incrementale pe loturi (nu conține funcția membră partial_fit) [20].

- **Perceptron**. Reprezintă un alt algoritm de clasificare simplu potrivit pentru testarea cu setul meu de date. Am folosit funcția standard din biblioteca Scikit-learn cu valorile implicite a toleranței (1e-3), numărului de iterații(1000) și al ratei de învățare(1e-3), cu excepția parametrului de amestecare al datelor după fiecare iterație pe care l-am setat fals pentru a obține performanțe mai bune [21].

- **Perceptron multi strat**. Pentru acest algoritm de ML am ales să desfășor testele pe două ahitecturi cu un număr de straturi ascunse diferit și implicit un număr de neuroni diferit: prima arhitectură cu un singur strat ascuns de 100 de neuroni (Figura 3.1), iar a doua arhitectură (Figura 3.2) formată din 3 straturi ascunse de dimensiuni diferite (900 de neuroni pe primul strat, 300 de neuroni pe al doilea strat și 30 de neuroni pe stratul numărul 3).

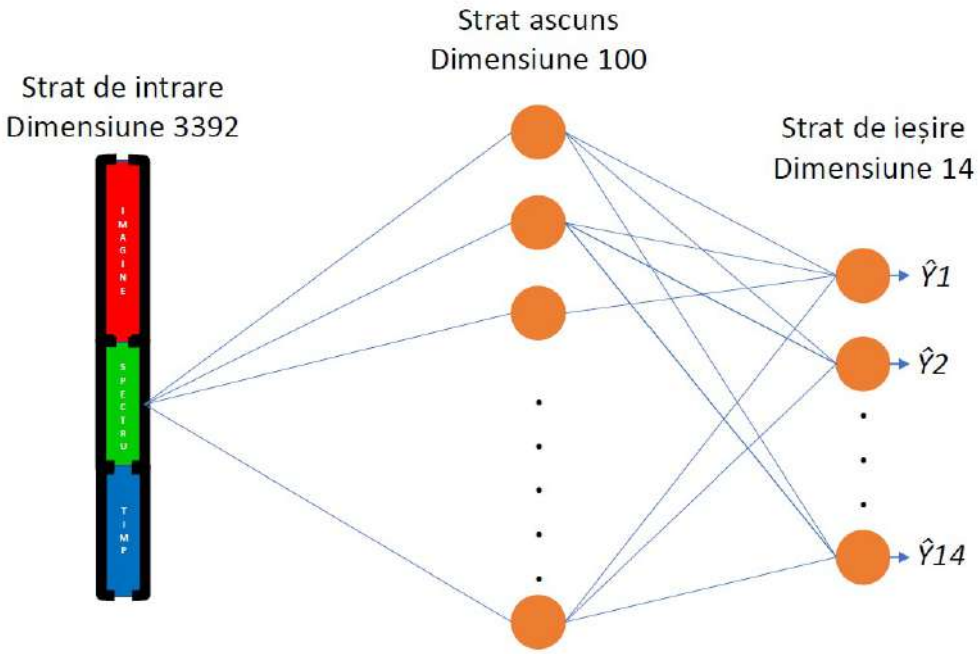


Figura 3.1: Arhitectura MLP cu un singur strat ascuns

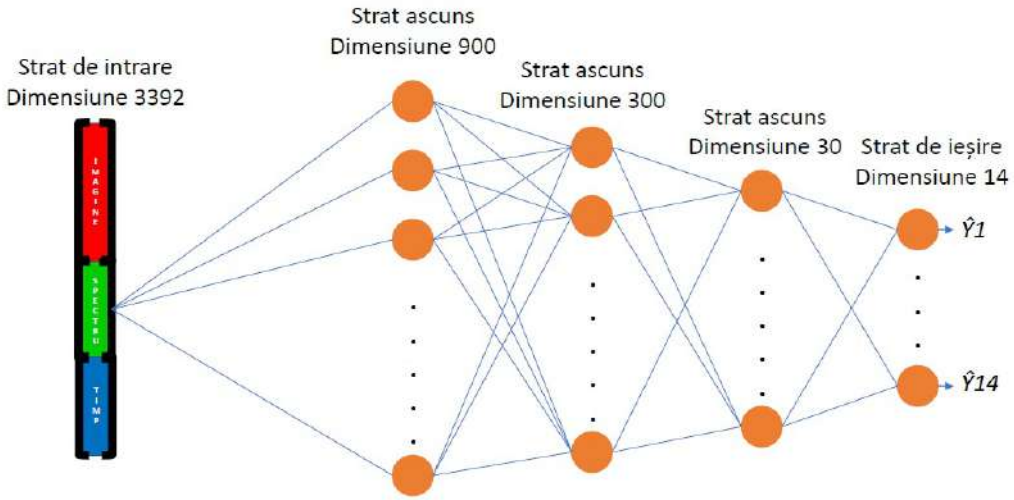


Figura 3.2: Arhitectura MLP cu multiple straturi ascunse

Am folosit funcția clasificatorului MLP cu parametrii impliciți din biblioteca Scikit-learn cu funția de activare de tip ReLU (Rectified Linear Unit) pentru o învățare mai rapidă și tipul de optimizare adam deoarece functionează cel mai bine pe seturi de date mari [22].

Primele experimente au fost realizate în timp ce lucram la preprocesarea de date în paralel. Aceste au fost realizate pe clasificatorul de tip perceptron și algoritmul celor mai apropiați vecini. Rezultatele obținute de la aceste inițiale experimente fiind foarte slabe (Tabela 3.1) m-au determinat să fac urmatorii pași de preprocesare a datelor și anume normalizarea datelor și PCA.

Clasificator	**Set de date neprelucrat**	**Set de date normalizat**
Perceptron	25% 22% 22% 39,64 s	52% 47% 47% 37,36 s
Algoritmul celor mai apropiați vecini	63% 34% 34% 608,05 s	42% 40% 40% 279,51 s

Tabela 3.1: Comparare rezultate inițiale cu rezultatele după aplicarea normalizării

Rezultatele notate în Tabela 3.1 sunt structurate astfel:

- Acuratețea modelului pe setul de antrenare
- Acuratețea modelului pe setul de testare
- Acuratețea modelului pe setul de validare
- Timpul de antrenare

Scopul acestor experimente este de a afla care algoritm de ML clasic și nivelul de preprocesare al datelor de intrare al acestui algoritm îmi clasifică cel mai bine setul de date SAU-SRB. Interesul principal este de a obține o acuratețe pe setul de test și validare cât mai mare. Al doilea criteriu de clasificare al rezultatelor este timpul de antrenare (dorim un timp cât mai mic de antrenare). Acuratețea clasificatorului pe setul de antrenare este mai puțin importantă în decizia alegerii celui mai bun algoritm. Acest parametru ne ajută mai mult în a vizualiza dacă algoritmul aplicat reușește să generalizeze setul de date sau învață doar setul de date.

Se observă din Tabela 3.1 că ambii algoritmi testați inițial au obținut o acuratețe mai mare pe setul de testare și validare normalizat, respectiv un timp de antrenare mai mic decât pe setul de date neprelucrat.

În continuare, am realizat experimentele și pentru ceilalți algoritmi de ML propuși pe baza de date normalizată și pe baza de date normalizată pe care am aplicat algoritmul PCA [23]. Rezultatele acestor experimente sunt prezentate în Tabela 3.2, unde N reprezintă numărul de componente păstrate după aplicarea algoritmului PCA.

Din matricea setului de date de dimensiune **3.392 X 86.513** (vezi Figura2.5) numărul de trăsături după aplicarea algoritmului PCA pe fiecare din cele 3 caracteristici (imaginea de împrăștiere, spectrul de fluorescență și timpul de răspuns fluorescent) este de 3 ori N deoarece păstrăm doar N componente din fiecare caracteristică. Astfel, dimensiunea matricei de date devine **(3N X 86.513)**.

Clasificator	Set de date neprelucrat	Set de date normat	Set de date normat +PCA (N=10)	Set de date normat +PCA (N=20)	Set de date normat +PCA (N=50)	Set de date normat +PCA (N=100)
Perceptron	25% 22% 22% 39,64 s	**52%** **47%** **47%** **37,36 s**	36% 36% 36% 0,30 s	31% 31% 31% 0,41 s	34% 34% 34% 0,92 s	33% 33% 33% 1,94 s
Bayes naiv	-	27% 27% 27% 28,62 s	**50%** **49%** **49%** **0,27 s**	50% 49% 49% 0,52 s	44% 43% 44% 1,15 s	40% 40% 40% 2,26 s
Algoritmul celor mai apropiați vecini	63% 34% 34% 608,05 s	42% 40% 40% 279,51 s	54% 51% 52% 108,1 s	**55%** **53%** **53%** **109,72 s**	54% 51% 52% 113,85 s	51% 49% 50% 123,22 s
MLP(100)	-	94% 60% 60% 2159,55 s	73% 70% 70% 154,79 s	**76%** **71%** **71%** **210,04 s**	80% 70% 70% 348,18 s	85% 68% 68% 367,10 s
MLP(900, 300, 30)	-	96% 64% 64% 6750,42 s	97% 68% 68% 506,16 s	**98%** **69%** **69%** **535,91 s**	98% 68% 69% 916,34 s	99% 68% 68% 1500,62 s

Tabela 3.2: Rezultate folosind algoritmi clasici de ML

Rezultatele notate în Tabela 3.2 au fost structurate la fel ca în Tabela 3.1, în ordinea următoare:

- Acuratețea modelului pe setul de antrenare
- Acuratețea modelului pe setul de testare
- Acuratețea modelului pe setul de validare
- Timpul de antrenare

De asemenea, am evidențiat în Tabela 3.2 cu galben cele mai bune rezultate pentru fiecare algoritm și cu albastru cel mai bun rezultat obținut folosind algoritmi de clasificare clasici de ML, acesta fiind atins de arhitectura MLP cu un singur strat ascuns pe setul de date normalizate pe care am aplicat PCA cu un număr de componente principale = 20.

Rezultatele din aceste experimente evidențiază foarte bine două concepte foarte importante din domeniul ML (Machine Learning), DL (Deep Learning) și anume conceptele de overfitting și underfitting.

Putem vedea conceptul de overfitting când comparăm rezultatele celor două arhitecturi MLP. Se observă că arhitectura MLP mai complexă se descurcă foarte bine pe setul de date de antrenare, acesta ajungând la performanțe de 99%, dar are performanțe mai slabe pe un set de date ne mai întâlnit deoarece acesta își actualizează ponderile pe toate detaliile din setul de date de intrare, inclusiv posibil zgomot. Astfel, modelul nu mai poate generaliza bine datele noi. Acesta fiind și motivul pentru care arhitectura MLP mai simplă a obținut un rezultat mai bun decât arhitectura MLP complexă, MLP(100) are o acuratețe de antrenare mai mică decât MLP(900, 300, 30), dar reușește să generalizeze mai bine seturile noi de date.

Fenomenul de underfitting se poate observa pe algoritmii de clasificare mai simpli de ML, precum perceptron și Bayes naiv, aceștia nereușind să extragă destulă informație din setul de date de antrenare și implicit au o acuratețe slabă pe seturi de date noi.

3.2 Clasificarea automată a polenului folosind rețele neurale convoluționale

Pentru a încerca să depășesc rezultatele obținute din studiul algoritmilor de clasificare clasici de ML, am realizat experimente și pe un model adânc de NN (Neural Network).

Deoarece două dintre caracteristicile de intrare și anume imaginea de împrăștiere, respectiv spectrul de fluorescență sunt reprezentate bidimensional ca și imagini, avem nevoie de un model care să țină cont și de distribuția spațială a caracteristicilor deoarece aceasta conține o mare parte din informația unei imagini.

Rețelele neurale convoluționale sunt foarte potrivite pentru prelucrarea și clasificarea imaginilor deoarece toate straturile unui CNN au multiple filtre convoluționale care scanează matricea de intrare și extrag informație asupra distribuției spațiale.

Am plecat de la o arhitectură convoluțională simplă, cunoscută în domeniul DL de clasificare al imaginilor și anume LeNet-5 și am realizat experimente pentru fiecare caracteristică separată în parte pentru a vedea câtă informație conține fiecare, urmând ca la final să realizez o arhitectură complexă care să se folosească de toate cele trei caracteristici.

3.2.1 Adaptarea LeNet-5 pentru clasificarea automată a polenului folosind imaginea de împrăștiere

După extragerea trăsăturii de interes din baza de date SAU-SRB și împărțirea acestui set de date nou care conține doar imagini de împrăștiere (86513 eșantioane), în set de antrenare 69210 eșantioane (80%) și set de testare 17303 eșantioane (20%), am realizat implementarea arhitecturii LeNet-5 și a codului de antrenare și testare în limbajul de programare Python cu ajutorul bibliotecii Pytorch și verificare funcționării corespunzătoare a acestuia pe un set de date foarte cunoscut precum MNIST descris în subcapitolul 1.2.3.

Pentru a obține performanțe cât mai buna pe setul de date propus, am realizat următorii pași de optimizare:

- **Adaptarea arhitecturii**. Prima etapă pe care am abordat-o a fost adaptarea arhitecturii pe setul meu de date (imagini de împrăștiere). Deoarece arhitectura LeNet-5 este cunoscută pentru rezultate foarte bune în clasificarea imaginilor în tonuri de gri pătrate de dimensiune 32 X 32, iar imaginile din setul meu de date sunt mai înguste pe axa ox (120 X 24), am decis să realizez o optimizare a dimensiunii filtrelor de convoluție aflate pe primele două straturi din NN și a bordării imaginii de intrare. Filtrul inițial de dimensiune 5 X 5 fără realizarea bordării intrării, eliminând foartă multă informație a contururilor din imaginea de împrăștiere.

  Pentru a păstra aceeași dimensiune la ieșirea straturilor convoluționale am setat constant pasul de deplasare (stride) al filtrului convoluțional ca fiind 1 (aceeași valoare din arhitectura LeNet-5), iar pentru a varia dimensiunea filtrului convoluțional (f) și dimensiunea bordării (p) astfel încât să nu micșorez dimensiunea ieșirii după parcurgerea unui strat convoluțional (same padding), m-am folosit de relația de calcul (1.4) pentru a implementa un script de evaluare automat al performanțelor diverselor arhitecturi cu dimensiunea filtrelor convoluționale și bordarea intrărilor variabilă.

  Am rulat experimentul inițial pentru un număr de epoci E=100, cu o rată de antrenare α=0,01, iar pentru o antrenare mai rapidă și diversificată am ales un număr de loturi L=1000 alese în ordine aleatoare în fiecare epocă din setul de antrenare cu ajutorul clasei membre Dataloader din biblioteca Pytorch [24].

Rezultatele pentru diverse dimensiuni ale filtrelor de convoluție se regăsesc în Tabela 3.3.

E=100; α=0,01; L=1000;

Dimensiunea filtrelor convoluționale	**f = 1**	**f = 3**	**f = 5**	**f = 7**	**f = 9**	**f = 11**
Dimensiunea bordării cu 0	**p = 0**	**p = 1**	**p = 2**	**p = 3**	**p = 4**	**p = 5**
Acuratețea pe setul de testare	17%	**49%**	37%	35%	32%	36%

Tabela 3.3: Rezultate CNN pentru diverse dimensiuni ale filtrelor de convoluție folosind imaginea de împrăștiere

În tabelele cu rezultate am evidențiat cea mai bună performanță obținută prin a marca cu albastru coloana corespunzătoare. După cum se poate observa în Tabela 3.3 am obținut cea mai mare acuratețe pe setul de test pentru dimensiunea filtrelor convoluționale egală cu 3 și dimensiunea bordării respective pentru a nu micșora dimensiunea ieșirii după parcurgerea unui strat convoluțional (same padding). Această dimensiune nou selectată fiind diferită față de dimensiunea filtrelor din arhitectura originală LeNet-5 care era de 5X5.

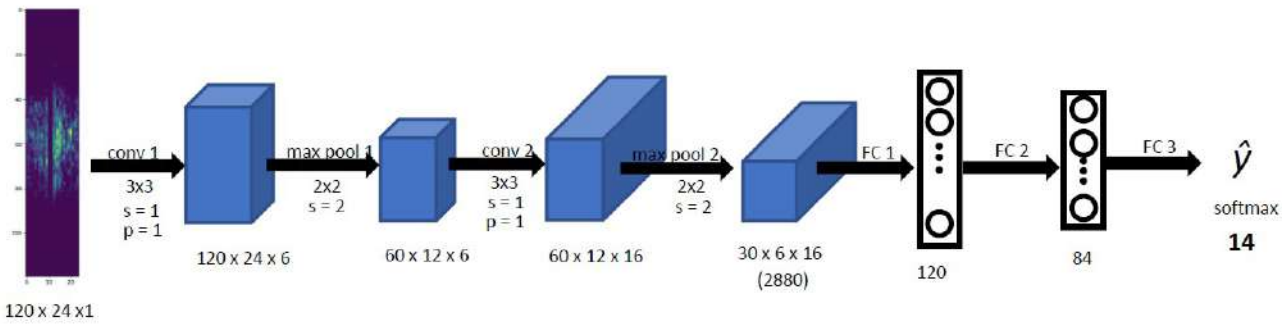


Figura 3.3: Reprezentarea arhitecturii LeNet-5 adaptată pentru imaginea de împrăștiere

În continuare am setat dimensiunea filtrului optim pe care am obținut-o și am trecut la următorul experiment în modelarea arhitecturii rețelei neurale pentru setul meu de date. Am încercat să modific arhitectura LeNet-5 (Figura 3.3) prin a mai adăuga un strat convoluțional (Figura 3.4).

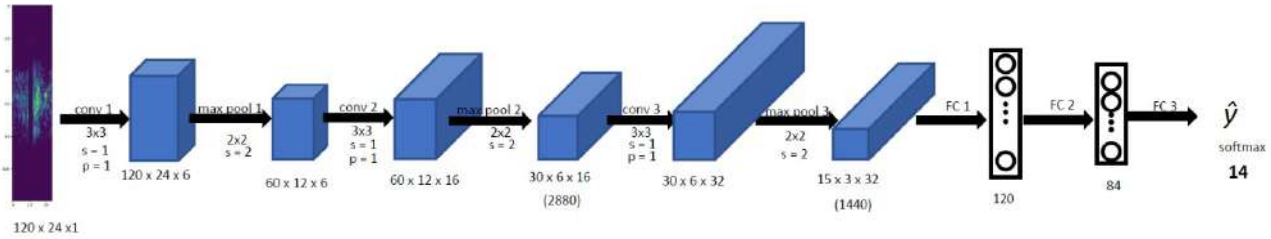


Figura 3.4: Reprezentarea arhitecturii LeNet-5 modificată cu un strat convoluțional adițional pentru imaginea de împrăștiere

După adăugarea stratului convoluțional în arhitectură am refăcut experimentul inițial pentru a observa rezultatele noii arhitecturi. Rezultatele obținute au fost similare cu cele din Tabela 3.3, cu excepția că timpul de rulare al experimentului a fost de aproximativ 6 ore. Această diferență a timpului de antrenare și faptul că acuratețea modelului nu se modifică m-a influențat să revin la arhitectura inițială prin eliminarea stratului convoluțional adăugat.

Deoarece modificarea parții convoluționale nu a adus nicio imbunătățire, am încercat să modific și straturile complet conectate ale rețelei (Figura 3.5).

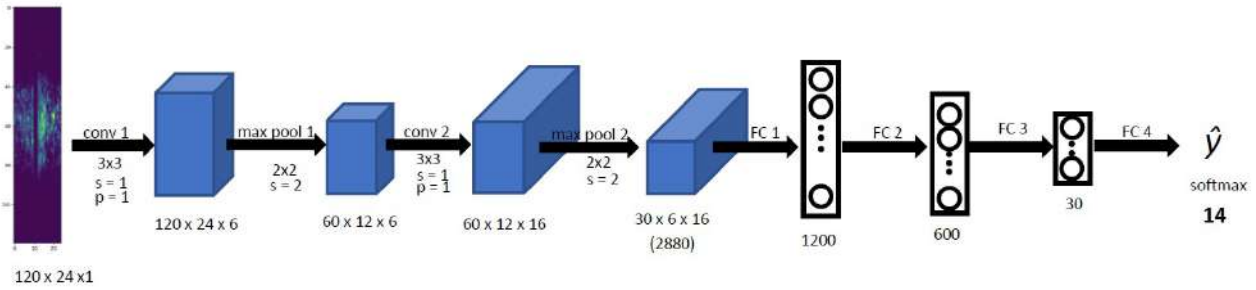


Figura 3.5: Reprezentarea arhitecturii LeNet-5 cu partea straturilor dense modificată pentru imaginea de împrăștiere

După reluarea experimentului inițial pe noua arhitectură performanța pentru fiecare dimensiune a filtrului nu depășea acuratețea de 17%. Motiv pentru care am revenit la arhitectura inițială și am continuat cu experimentele de optimizare a hiperparametrilor.

- **Optimizarea hiperparametrilor/Adaptarea antrenării**. Deoarece rata de antrenare aleasă inițial este destul de mare, există șansa ca algoritmul să sară peste minimul dorit. De asemenea, dacă selectăm o rată de învățare prea mică algoritmul o să ajungă în minim din foarte mulți pași și implicit într-un timp crescut, de aceea pentru acest experiment am crescut numărul de epoci.

Rezultatele pentru diverse rate de antrenare se regăsesc în Tabela 3.4.

E=600; f=3, p=1; L=1000;

Rata de antrenare	**0.01**	**0.001**	**1e-4**	**1e-5**
Acuratețea pe setul de testare	49%	50%	**51%**	32%

Tabela 3.4: Rezultate CNN pentru diverse rate de antrenare folosind imaginea de împrăștiere

În urma acestui experiment am reușit să îmbunătățesc performanța modelului cu 3% prin micșorarea ratei de învățare.

Pentru următorul experiment am modificat valoarea noii rate de antrenare cu 1e-4 și am realizat o căutare a dimenisunii mini loturilor optimă (L). Această dimensiune reprezintă numărul de eșantioane procesate după care modelul își actualizează ponderile și poate afecta semnificativ performanța și timpul de antrenare. Loturile de antrenare pot avea orice dimensiune între 1 și dimensiunea întregului set de antrenare. Pentru experimentul acesta am decis să verific plecând de la dimensiunea 256 până la 16384 trecând prin puterile lui 2. De asemenea, am evaluat și performanțele modelului fără a împărții setul de antrenare în mini loturi.

Rezultatele pentru diverse dimensiuni ale loturilor de antrenare se regăsesc în Tabela 3.5.

E=1000; f=3, p=1; α=1e-4;

Dimensiunea lotului de antrenare	**256**	**512**	**1024**	**2048**	**4096**	**8192**	**16384**	**întreg setul de antrenare (69210)**
Acuratețea pe setul de testare	55%	**56%**	54%	54%	52%	51%	48%	44%

Tabela 3.5: Rezultate CNN pentru diverse dimensiuni ale loturilor de antrenare folosind imaginea de împrăștiere

În urma acestui experiment am reușit să imbunătățesc acuratețea modelului CNN care se folosește doar de eșantioanele imaginilor de împrăștiere de la 51% până la 56%, folosind loturi de dimensiune 512 în loc de loturi de antrenare de dimensiune 1000 aleasă inițial.

- **Corectarea implementării**. Pentru a obține probabilitățile fiecărei clase la ieșirea rețelei am folosit funcția de activare softmax (aceasta reprezentând extesia funcției sigmoidale pentru mai mult de două clase).

  O greșeală comună în folosirea bibliotecii Pytorch pentru construirea unei rețele neurale este folosirea funcției softmax pe ieșirea rețelei în timp ce folosești criteriul de optimizare CrossEntropyLoss [25], deoarece acesta aplică implicit funcția de activare softmax pe ieșirea rețelei.

  Astfel, modelul aplică de două ori funcția softmax pe ieșirea rețelei fapt care contribuie la o antrenare mai lentă și valori ridicate în calculul erorii.

  După corectarea implementării algoritmului am reantrenat rețeaua folosind arhitectura și hiperparametrii optimi aflați în urma experimentelor și am obținut următoarele rezultate.

  Rezultatele după corectarea implementării algoritmului de învățare se regăsesc în Tabela 3.6.

	Implementarea inițială	**Implementarea corectată**
Acuratețea modelului pe setul de test	56%	**57%**
Numărul de epoci	1000	**200**
Timpul de antrenare	4 ore și 24 minute	**56 minute**

Tabela 3.6: Rezultate după corectarea implementării algoritmului de învățare folosind imaginea de împrăștiere

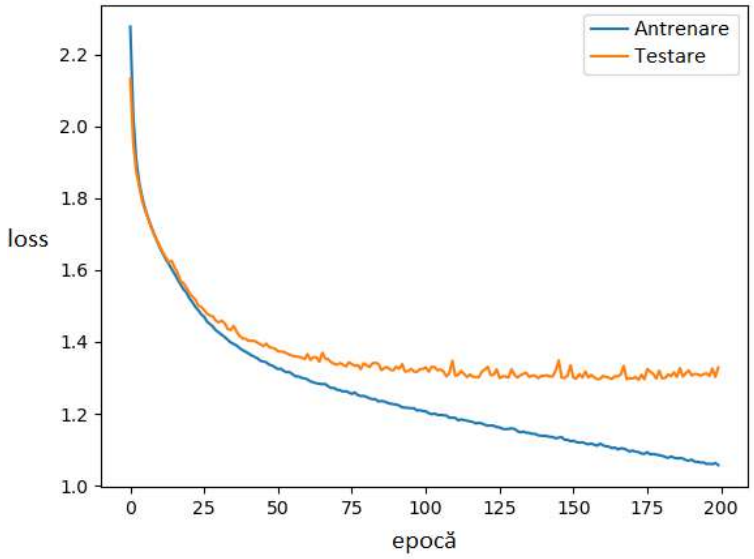


Figura 3.6: Vizualizarea funcției loss pentru antrenarea, respectiv testarea modelului folosind imaginea de împrăștiere

Se poate observa în Figura 3.6 faptul că de la epoca 50 loss-ul modelului pe baza de testare începe să scadă foarte greu. Dacă lăsam modelul să continuie antrenarea pentru mai multe epoci de 200, acuratețea modelului pe setul de test începea să scadă încet deoarece modelul supraînvăța pe setul de antrenare.

Ultimul modelul a fost apoi salvat pentru a-l utiliza în arhitectura finală multi-intrare.

3.2.2 Adaptarea LeNet-5 pentru clasificarea automată a polenului folosind spectrul de fluorescență

Pentru adaptarea arhitecturii rețelei convoluționale folosind spectrul de fluorescență am inițializat hiperparametrii cu cei optimi găsiți în urma experimentelor precedente și am repetat experimentele pe setul de date format doar din spectrul de fluorescență.

- **Adaptarea arhitecturii**.

  Rezultatele pentru diverse dimensiuni ale filtrelor de convoluție se regăsesc în Tabela 3.7.

  E=1000; α=1e-4; L=256;

Dimensiunea filtrelor convoluționale	**f = 1**	**f = 3**	**f = 5**	**f = 7**	**f = 9**	**f = 11**
Dimensiunea bordării cu 0	**p = 0**	**p = 1**	**p = 2**	**p = 3**	**p = 4**	**p = 5**
Acuratețea pe setul de testare	53%	56%	56%	**57%**	**57%**	56%

Tabela 3.7: Rezultate CNN pentru diverse dimensiuni ale filtrelor de convoluție folosind spectrul de fluorescență

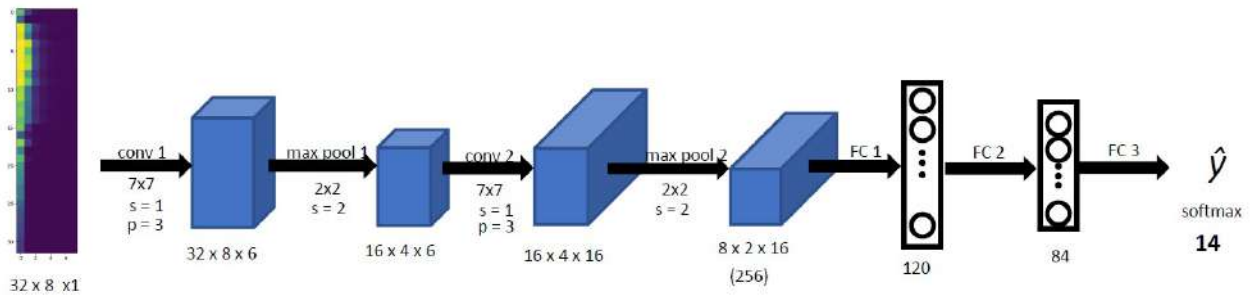


Figura 3.7: Reprezentarea arhitecturii LeNet-5 adaptată pentru spectrul de fluorecență

- **Optimizarea hiperparametrilor/Adaptarea antrenării.**

  Rezultatele pentru diverse rate de antrenare se regăsesc în Tabela 3.8.

  E=1000; f=7, p=3; L=256;

Rata de antrenare	**0.01**	**0.001**	**1e-4**	**1e-5**
Acuratețea pe setul de testare	18%	56%	**57%**	52%

Tabela 3.8: Rezultate CNN pentru diverse rate de antrenare folosind spectrul de fluorescență

Rezultatele pentru diverse dimensiuni ale loturilor de antrenare se regăsesc în Tabela 3.9.

E=1000; f=7, p=3; α=1e-4;

Dimensiunea lotului de antrenare	**256**	**512**	**1024**	**2048**	**4096**	**8192**	**16384**	**întreg setul de antrenare (69210)**
Acuratețea pe setul de testare	**57%**	56%	55%	54%	52%	52%	40%	29%

Tabela 3.9: Rezultate CNN pentru diverse dimensiuni ale loturilor de antrenare folosind spectrul de fluorescență

- **Corectarea implementării.**

  Rezultatele după corectarea implementării algoritmului de învățare se regăsesc în Tabela 3.10.

	Implementarea inițială	**Implementarea corectată**
Acuratețea modelului pe setul de test	57%	**59%**
Numărul de epoci	1000	**500**
Timpul de antrenare	42 minute	**20 minute**

Tabela 3.10: Rezultate după corectarea implementării algoritmului de învățare folosind spectrul de fluorescență

Modelul cu performanța cea mai mare obținut în urma experimentelor pe baza de date formată doar din spectrul de fluorescență a fost apoi salvat pentru a fi utilizat în arhitectura finală multi-intrare. Reprezentarea funcției loss a acestui model se regăsește în Figura 3.8.

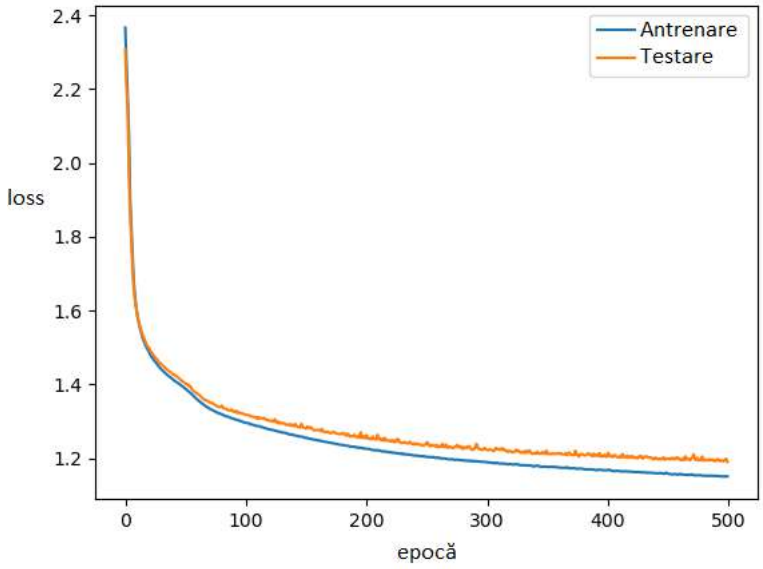


Figura 3.8: Vizualizarea funcției loss pentru antrenarea, respectiv testarea modelului folosind doar spectrul de fluorescență

3.2.3 Adaptarea LeNet-5 pentru clasificarea automată a polenului folosind timpul de răspuns fluorescent

Pentru adaptarea arhitecturii rețelei convoluționale folosind timpul de răspuns fluorescent am inițializat hiperparametrii cu cei optimi găsiți în urma experimentelor precedente și am repetat experimentele pe setul de date format doar din timpul de răspuns fluorescent.

- **Adaptarea arhitecturii**

  Rezultatele pentru diverse dimensiuni ale filtrelor de convoluție se regăsesc în Tabela 3.11.

  E=1000; α=1e-4; L=256;

Dimensiunea filtrelor convoluționale	**f = 1**	**f = 3**	**f = 5**	**f = 7**	**f = 9**	**f = 11**
Dimensiunea bordării cu 0	**p = 0**	**p = 1**	**p = 2**	**p = 3**	**p = 4**	**p = 5**
Acuratețea pe setul de testare	55%	61%	62%	60%	**63%**	62%

Tabela 3.11: Rezultate CNN pentru diverse dimensiuni ale filtrelor de convoluție folosind timpul de răspuns fluorescent

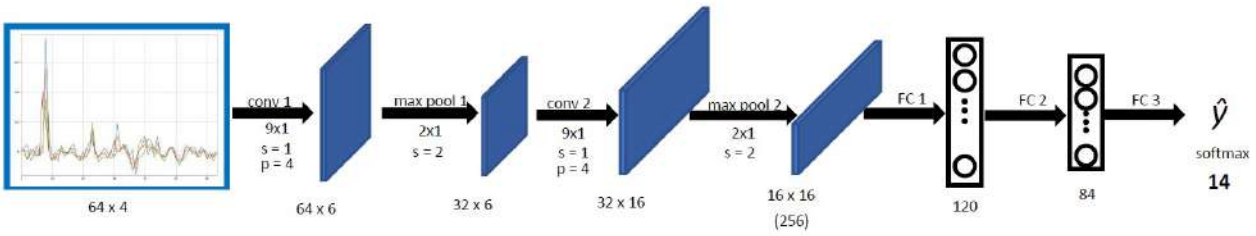


Figura 3.9: Reprezentarea arhitecturii LeNet-5 adaptată pentru timpul de răspuns fluorescent

Pentru caracteristica de timp de răspuns fluorescent am folosit o convoluție unidimensională, iar pentru imaginile de împrăștiere și spectrele de fluorescență am aplicat o convoluție bidimensională.

- **Optimizarea hiperparametrilor/Adaptarea antrenării**

  Rezultatele pentru diverse rate de antrenare se regăsesc în Tabela 3.12.

  E=1000; f=9, p=4; L=256;

Rata de antrenare	**0.01**	**0.001**	**1e-4**	**1e-5**
Acuratețea pe setul de testare	54%	61%	**63%**	47%

Tabela 3.12: Rezultate CNN pentru diverse rate de antrenare folosind timpul de răspuns fluorescent

Rezultatele pentru diverse dimensiuni ale loturilor de antrenare se regăsesc în Tabela 3.13.

E=1000; f=9, p=4; α=1e-4;

Dimensiunea lotului de antrenare	**256**	**512**	**1024**	**2048**	**4096**	**8192**	**16384**	**întreg setul de antrenare (69210)**
Acuratețea pe setul de testare	**63%**	**63%**	61%	48%	55%	38%	34%	30%

Tabela 3.13: Rezultate CNN pentru diverse dimensiuni ale loturilor de antrenare folosind timpul de răspuns fluorescent

- **Corectarea implementării**

  Rezultatele după corectarea implementării algoritmului de învățare se regăsesc în Tabela 3.14.

	Implementarea inițială	**Implementarea corectată**
Acuratețea modelului pe setul de test	63%	**66%**
Numărul de epoci	1000	**500**
Timpul de antrenare	35 minute	**12 minute**

Tabela 3.14: Rezultate după corectarea implementării algoritmului de învățare folosind timpul de răspuns fluorescent

Modelul cu performanța cea mai mare obținut în urma experimentelor pe baza de date formată doar din timpul de răspuns fluorescent a fost apoi salvat pentru a fi utilizat în arhitectura finală multi-intrare. Reprezentarea funcției loss a acestui model se regăsește în Figura 3.10.

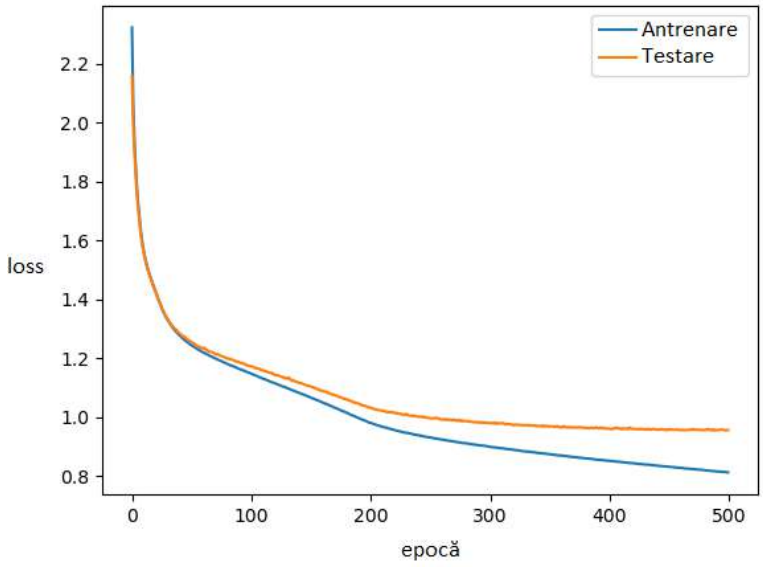


Figura 3.10: Vizualizarea funcției loss pentru antrenarea, respectiv testarea modelului folosind doar timpul de răspuns fluorescent

3.2.4 Arhitectura CNN multi-intrare

Arhitectura CNN multi-intrare se folosește de toate cele 3 caracteristici oferite de setul de date SAU-SRB. Am încercat două arhitecturi diferite multi-intrare:

- **Arhitectură CNN multi-input bazată pe rețelele deja antrenate obținute în urma experimentelor precedente**.

  Cele 3 rețele salvate anterior pentru fiecare caracteristică în parte au fost încărcate în același script pentru a realiza următoarea arhitectură prezentata în Figura 3.11.

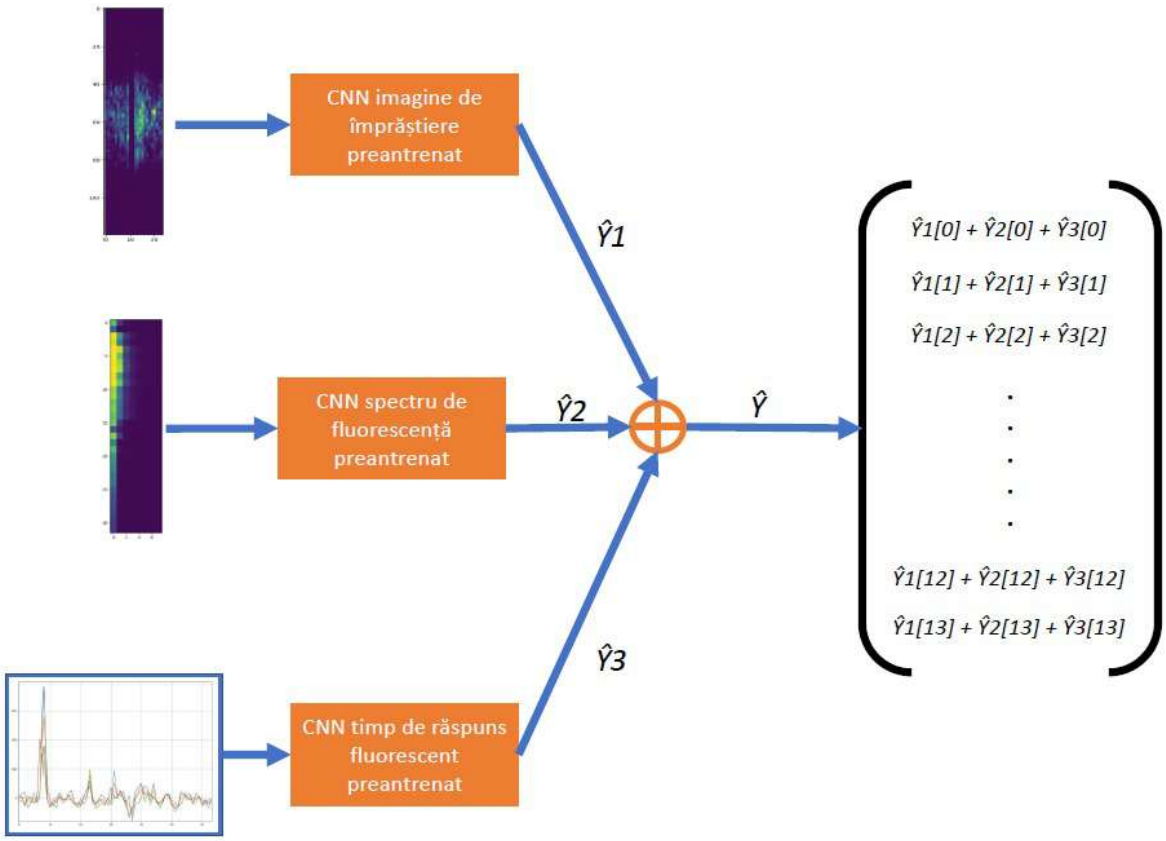


Figura 3.11: Reprezentarea arhitecturii LeNet-5 multi-intrare bazată pe rețelele preantrenate și adaptată pentru cele 3 caracteristici

Vectorii de la ieșirea fiecărei rețele neurale conțin probabilitatea de apartenență la cele 14 clase de polen. Fiecare rețea având o predicție separată cu acuratețile următoare:

- **57% CNN imagine de împrăștiere**
- **59% CNN spectru de fluorescență**
- **66% CNN timp de răspuns fluorescent**

Realizând o simplă sumă peste acești vectori de ieșire de la fiecare rețea obținem o noua predicție pentru situațiile în care probabilitățile de apartenență la cele 14 clase sunt apropiate (Rețeaua neurală nu poate generaliza cu o precizie ridicată), astfel obținând o acuratețe mai mare deoarece folosim informațiile oferite de toate cele 3 caracteristici.

Acuratețea obținută pentru această arhitectură = 75%

- **Arhitectură CNN multi-input bazată pe arhitecturile obținute în urma experimentelor precedente**.

  Față de arhitectura multi-intrare precedentă (Figura 3.11), această rețea (Figura 3.12) nu folosește modelele deja antrenare ci doar arhitecturile și hiperparametrii optimi găsiți în urma experimentelor. Aceasta având nevoie de un nou preces de antrenare pentru optimizarea ponderilor.

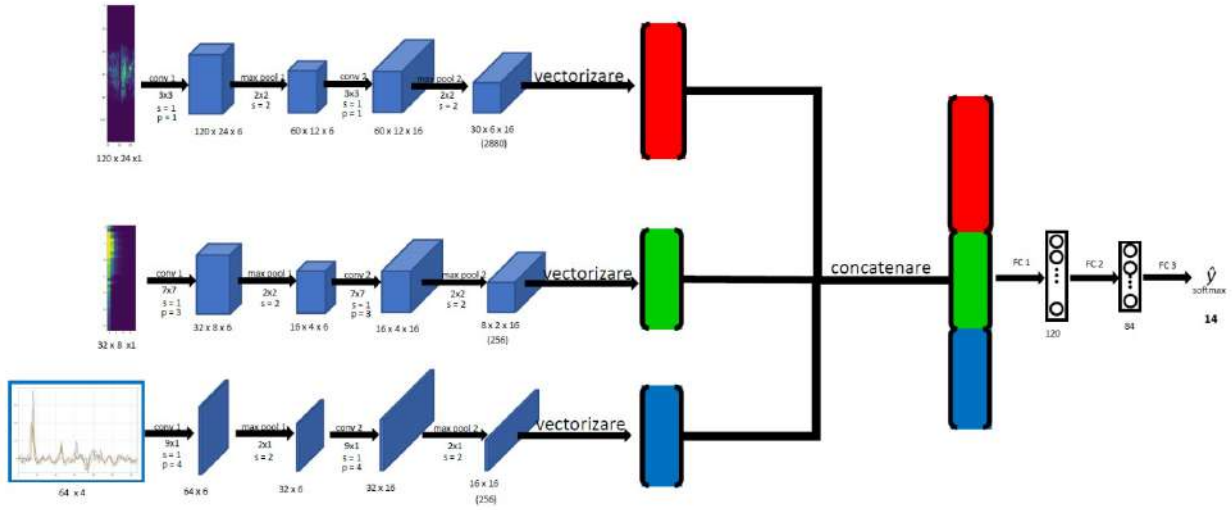


Figura 3.12: Reprezentarea arhitecturii LeNet-5 multi-intrare bazată pe arhitecturile precedente și adaptată pentru cele 3 caracteristici

Modelul preia ieșirea din partea convoluțională a fiecărei caracteristici și le vectorizează pentru ca mai apoi să concateneze acesti 3 vectori. Vectorul format din concatenarea ieșirilor este dat mai departe ca input în partea straturilor complet conectate a rețelei pentru a calcula probabilitățile de apartenență la cele 14 clase de polen.

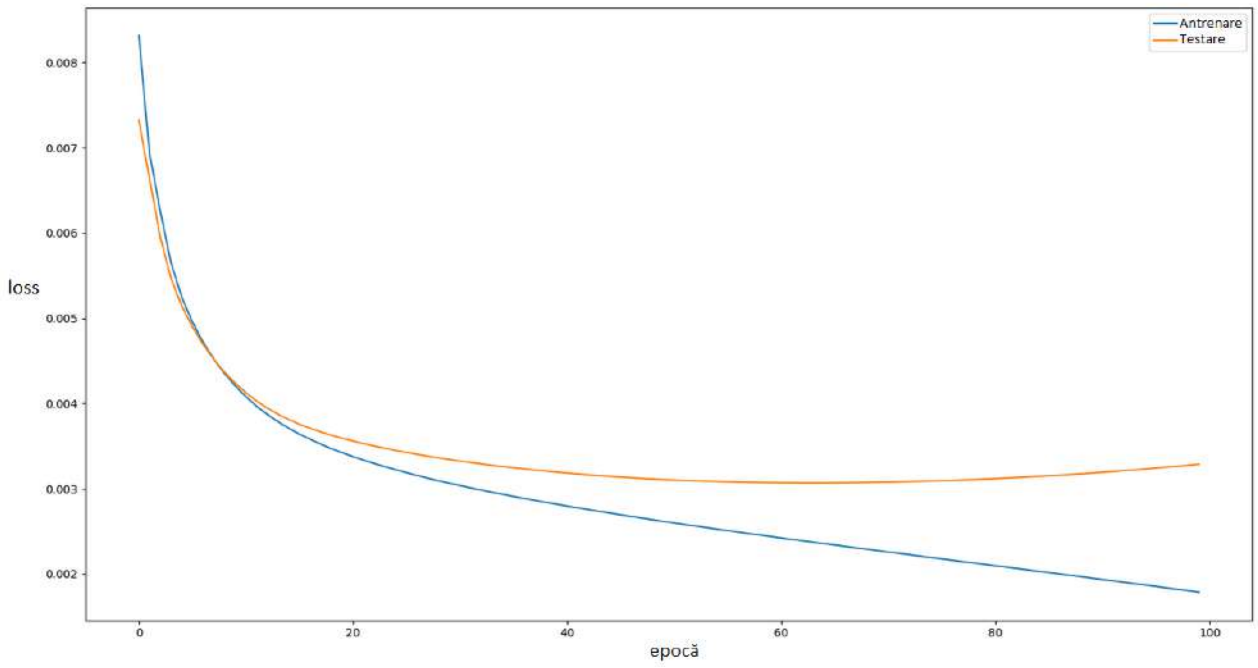


Figura 3.13: Vizualizarea funcției loss pentru antrenarea, respectiv testarea arhitecturii multi-intrare

Din Figura 3.13 se poate observa că modelul începe să supraînvețe pe setul de antrenare deoarece funcția loss continuă să scadă pe setul de antrenare, iar cea pe setul de testare începe să crească, fapt care duce la o acuratețe mai slabă pe setul de testare. Pentru rezolvarea acestei probleme se poate utiliza metoda de oprire a antrenării în momentul în care funcția loss începe să crească (early stopping).

Acuratețea obținută pentru această arhitectură = 73%

Așteptarea era ca pentru această arhitectură (Figura 3.12) să obțin perfomanțe mai bune decât în experimentele cu arhitectura din Figura 3.11.

Consider că performanțele mai slabe sunt datorate părții dense a arhitecturii întrucât arhitectura din figura 3.11 are aceeași arhitectură densă pentru fiecare caracteristică, iar acest model (Figura 3.12) prezintă doar o singură parte densă pentru toate cele 3 caracteristici concatenate.

Deși această arhitectură are o performanță mai slabă decât arhitectura multi-input precedent prezentată în lucrare, acest model poate fi în continuare îmbunătățit prin optimizarea hiperparametrilor.

3.3 Sumar rezultate

Pentru a sumariza toate rezultatele obținute în această lucrare am realizat Tabela 3.15.

Algoritmi	Model	Acuratețe	Timp de antrenare
de clasificare clasici (ML)	**Perceptron**	47%	37 secunde
	Bayes naiv	49%	1 secunde
	Cei mai apropiați k vecini	53%	2 minute
	MLP (900,300,30)	69%	9 minute
	MLP (100)	71%	4 minute
de învățare profundă (DL)	**LeNet-5 multi-intrare**	73%	36 minute
	LeNet-5 multi-intrare sumă ieșire	**75%**	56 minute

Tabela 3.15: Sumar rezultate

Chiar dacă performanța maximă atinsă nu este una ideală, se poate observa că în urma experimentelor multiple realizate pe parcursul lucrării performanțele cresc semnificativ.

Performanța cea mai mare fiind atinsă de modelul LeNet-5 multi-intrare care realizează suma pe ieșirile rețelelor preantrenate. Am ales timpul de antrenare pentru acest model ca fiind cel mai mare timp de antrenare al celor 3 rețele preantrenate, deoarece acestea pot fi antrenate în paralel.

Comparând rezultatele obținute de către mine cu alte lucrări de clasificare a polenului pe aceeași bază de date precum proiectul de cercetare [15] din 2019, care a obținut o performanță maximă de 73%, se poate observa că am reușit să măresc puțin performanțele folosind o arhitectură convoluțională mai simplă.

Concluzii

Concluzii generale

Clasificarea și studiul polenului reprezintă o sarcină cu un domeniu de aplicabilitate larg în diverse industrii. Aceasta poate fi abordată prin mai multe moduri, însă rezolvarea utilizând algoritmi de clasificare clasici și algoritmi de învățare profundă reprezintă abordarea cu cel mai mare potențial, chiar dacă aceasta este foarte dificilă.

Principala provocare în ceea ce privește dezvoltarea unui asemenea clasificator este căutarea arhitecturii și hiperparametrilor optimi, aceasta rezolvându-se prin multiple experimente care se pot întinde pe perioade destul de lungi de timp în funcție de puterea de calcul și complexitatea modelului studiat.

De asemenea, alte provocări pot fi întâlnite și înaintea procesului de antrenare și testare, și anume: colectarea unui set de date îndeajuns de mare, filtrarea și preprocesarea bazei de date pentru asigurarea performanțelor ridicate.

Scopul acestei lucrări a fost studiul, aplicarea și adaptarea diverșilor algoritmi de clasificare clasici și învățare profundă pe setul de date SAU-SRB. Pentru realizarea acestui scop a fost mai întâi nevoie de înțelegerea setului de date, filtrarea și preprocesarea acetuia. Deși performanțele obținute în clasificarea polenului nu sunt ideale, se poate observa că pe parcursul lucrării performanța și complexitatea modelelor a crescut.

Contribuții personale

- Familiarizarea cu setul de date SAU-SRB
- Filtrarea și prelucrarea setului de date
- Aplicarea algoritmilor de preprocesare pe setul de date (PCA)
- Studiul algoritmilor de clasificare clasici (perceptron, Bayes naiv, cei mai apropiați k vecini, proces gaussian, MLP)
- Studiul algoritmilor de învățare profundă ce pot fi utilizați pentru clasificarea polenului (CNN) și optimizarea arhitecturilor și hiperparametrilor pe problema propusă
- Utilizarea algoritmilor și implementarea experimentelor în limbajul de programare Python cu ajutorul diverselor biblioteci
- Realizarea experimentelor și interpretarea rezultatelor pentru îmbunătățirea perfomanțelor

Dezvoltări viitoare

- Îmbunătățirea modului de căutare a hiperparametrilor. De exemplu, implementarea unei rate de învățare variabile pentru o precizie mai mare în găsirea minimului.
- Optimizarea hiperparametrilor arhitecturii CNN multi-input bazată pe arhitecturile obținute în urma experimentelor precedente
- Adaptarea unei arhitecturi CNN mai complexă decât LeNet-5 (AlexNet)

Bibliografie

[1] VR Dhawale, JA Tidke, and SV Dudul. Efficient classification of pollen grains using computational intelligence approach. In *International Conference for Convergence for Technology-2014*, pages 1–6. IEEE, 2014.

[2] Insects and Pollinators. https://www.nrcs.usda.gov/wps/portal/nrcs/main/national/plantsanimals/pollinate/. Accesat 17 iunie 2021.

[3] The importance of bees. https://www.worldbeeday.org/en/about/the-importance-of-bees.html. Accesat 17 iunie 2021.

[4] V. Neagoe. *Clasificatori clasici*. Curs RFIA, 2018, paginile 1-4.

[5] Harikumar Rajaguru, Sunil Kumar Prabhakar. *KNN Classifier and K-means clustering for Robust Classification of Epilepsy from EEG Signals*. Anchor Academic Publishing, 2017, paginile 31-33.

[6] Scikit-learn. Gaussian Processes. https://scikit-learn.org/stable/modules/gaussian_process.html#gaussian-process-classification-gpc. Accesat 18 aprilie 2021.

[7] Datawow. Interns Explain Basic Neural Network. https://datawow.io/blogs/interns-explain-basic-neural-network. Accesat 28 iunie 2021.

[8] Andrew Ng. Convolutional Neural Networks. https://www.coursera.org/learn/convolutional-neural-networks/lecture/Xv7B5/why-convolutions. Accesat 28 iunie 2021.

[9] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W.Hubbard, L.D. Jackel. *Backpropagation Applied to Handwritten Zip Code Recognition*. Massachusetts Institute of Technology, 1989, paginile 1-11.

[10] Josef Steppan. MnistExamples. https://commons.wikimedia.org/wiki/File:MnistExamples.png. Accesat 29 iunie 2021.

[11] Víctor Sevillano, Katherine Holt, José L, Aznarte. Precise automatic classification of 46 different pollen types with convolutional neural networks. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0229751. Accesat 21 iunie 2021.

[12] Paul Comtois. The experimental research of Charles H. Blackley. https://link.springer.com/article/10.1007%2FBF02136147. Accesat 21 iunie 2021.

[13] J. M. HIRST. AN AUTOMATIC VOLUMETRIC SPORE TRAP. https://doi.org/10.1111/j.1744-7348.1952.tb00904.x. Accesat 21 iunie 2021.

[14] Mihai Boldeanu, Horia Cucu, Corneliu Burileanu, Luminița Mărmureanu. *Automatic pollen classification using convolutional neural networks*. ETTI, University Politehnica of Bucharest, Bucharest, Romania.

[15] Ingrida Šauliene. Automatic pollen recognition with the rapid-e particle counter:the first-level procedure, experience and next steps. In *vol. 12*, pages 3436–3452, 2019.

[16] Mihai Ciuc, Constantin Vertan. *Prelucrarea statistică a semnalelor*. Editura MatrixRom, 2005, paginile 136-140.

[17] Scikit-learn. Gaussian Naive Bayes. `https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB`. Accesat 25 aprilie 2021.

[18] Scikit-learn. K Neighbors Classifier. `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier.kneighbors`. Accesat 25 aprilie 2021.

[19] Scikit-learn. Gaussian Process Classifier. `https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.GaussianProcessClassifier.html#sklearn.gaussian_process.GaussianProcessClassifier`. Accesat 25 aprilie 2021.

[20] Scikit-learn. Incremental learning. `https://scikit-learn.org/stable/computing/scaling_strategies.html#incremental-learning`. Accesat 25 aprilie 2021.

[21] Scikit-learn. Perceptron. `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html#sklearn.linear_model.Perceptron`. Accesat 24 aprilie 2021.

[22] Scikit-learn. MLP Classifier. `https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier`. Accesat 26 aprilie 2021.

[23] Scikit-learn. PCA. `https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html`. Accesat 5 mai 2021.

[24] Pytorch. Dataloader. `https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader`. Accesat 9 mai 2021.

[25] Pytorch. CROSSENTROPYLOSS. `https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html`. Accesat 18 iunie 2021.