



UNIVERSITATEA DIN CRAIOVA
FACULTATEA DE AUTOMATICĂ, CALCULATOARE ȘI
ELECTRONICĂ
DEPARTAMENTUL DE CALCULATOARE ȘI TEHNOLOGIA
INFORMAȚIEI



PROIECT DE DISERTAȚIE

Gabriela-Loredana Dinu

COORDONATOR ȘTIINȚIFIC

Prof. Dr. Ing. Liana Stănescu

Iulie, 2022

CRAIOVA



UNIVERSITATEA DIN CRAIOVA
FACULTATEA DE AUTOMATICĂ, CALCULATOARE ȘI
ELECTRONICĂ
DEPARTAMENTUL DE CALCULATOARE ȘI TEHNOLOGIA
INFORMAȚIEI



Studiu experimental asupra unor metode de clasificare a imaginilor

Gabriela-Loredana Dinu

COORDONATOR ȘTIINȚIFIC

Prof. Dr. Ing. Liana Stănescu

Iulie, 2022

CRAIOVA

„Învățătura este o comoară care își urmează stăpânul pretutindeni.”

Proverb popular

DECLARAȚIE DE ORIGINALITATE

Subsemnatul Gabriela-Loredana Dinu, student la specializarea Inginerie Software din cadrul Facultății de Automatică, Calculatoare și Electronică a Universității din Craiova, certific prin prezenta că am luat la cunoștință de cele prezentate mai jos și că îmi asum, în acest context, originalitatea proiectului meu de licență:

- cu titlul „Studiu experimental asupra unor metode de clasificare a imaginilor”
- coordonată de Prof. Dr. Ing. Liana Stănescu
- prezentată în sesiunea Iulie 2022

La elaborarea proiectului de licență, se consideră plagiat una dintre următoarele acțiuni:

- reproducerea exactă a cuvintelor unui alt autor, dintr-o altă lucrare, în limba română sau prin traducere dintr-o altă limbă, dacă se omit ghilimele și referința precisă,
- redarea cu alte cuvinte, reformularea prin cuvinte proprii sau rezumarea ideilor din alte lucrări, dacă nu se indică sursa bibliografică,
- prezentarea unor date experimentale obținute sau a unor aplicații realizate de alți autori fără menționarea corectă a acestor surse,
- însușirea totală sau parțială a unei lucrări în care regulile de mai sus sunt respectate, dar care are alt autor.

Pentru evitarea acestor situații neplăcute se recomandă:

- plasarea între ghilimele a citatelor directe și indicarea referinței într-o listă corespunzătoare la sfârșitul lucrării,
- indicarea în text a reformulării unei idei, opinii sau teorii și corespunzător în lista de referințe a sursei originale de la care s-a făcut preluarea,
- precizarea sursei de la care s-au preluat date experimentale, descrieri tehnice, figuri, imagini, statistici, tabele et caetera,
- precizarea referințelor poate fi omisă dacă se folosesc informații sau teorii arhicunoscute, a căror paternitate este unanim cunoscută și acceptată.

Data,

24/06/2022

Semnătura candidatului,



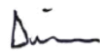


UNIVERSITATEA DIN CRAIOVA
Facultatea de Automatică, Calculatoare și Electronică
Departamentul de Calculatoare și Tehnologia Informației

Aprobat la data de
.....
Șef de departament,
Prof. dr. ing.
Marius BREZOVAN

PROIECTUL DE DIPLOMĂ

Numele și prenumele studentului/-ei:	Dinu Gabriela-Loredana
Enunțul temei:	Studiu experimental asupra unor metode de clasificare a imaginilor
Datele de pornire:	Proiectarea unei aplicații ce conține minim două metode de clasificare de imagini și compararea acestora din punct de vedere al timpului de învățare și a altor metrici precum acuratețe, precizie, reapel și specificitate de clasificare, urmată de implementarea unui program de clasificare a imaginilor aparținând unor clase predefinite pe baza algoritmilor analizați.
Conținutul proiectului:	<ol style="list-style-type: none">1. Introducere2. Etapa de analiză3. Despre algoritmii utilizați4. Aspecte teoretice folosite în proiect5. Despre proiect6. Concluzii7. Manual de utilizare8. Bibliografie9. Referințe Web A. Anexe cu codul sursă
Material grafic obligatoriu:	Diagrame, Scheme, Poze, Grafice, Tabele
Consultații:	Periodice
Conducătorul științific (titlul, nume și prenume, semnătura):	Prof. Dr. Ing. Liana Stănescu
Data eliberării temei:	15.10.2021

Termenul estimat de predare a proiectului:	25.06.2022
Data predării proiectului de către student și semnătura acestuia:	24.06.2022 



UNIVERSITATEA DIN CRAIOVA
Facultatea de Automatică, Calculatoare și Electronică
Departamentul de Calculatoare și Tehnologia Informației

REFERATUL CONDUCĂTORULUI ȘTIINȚIFIC

Numele și prenumele candidatului/-ei: Dinu Gabriela-Loredana
Specializarea: Inginerie Software
Titlul proiectului: Studiu experimental asupra unor metode de clasificare a imaginilor
Locația în care s-a realizat practica de documentare (se bifează una sau mai multe din opțiunile din dreapta):
În facultate ☒
În producție ☐
În cercetare ☐
Altă locație: *[se detaliază]*

În urma analizei lucrării candidatului au fost constatate următoarele:

Nivelul documentării		Insuficient <input type="checkbox"/>	Satisfăcător <input type="checkbox"/>	Bine <input type="checkbox"/>	Foarte bine <input checked="" type="checkbox"/>
Tipul proiectului		Cercetare <input type="checkbox"/>	Proiectare <input type="checkbox"/>	Realizare practică <input type="checkbox"/>	Altul <i>[se detaliază]</i>
Aparatul matematic utilizat		Simplu <input type="checkbox"/>	Mediu <input type="checkbox"/>	Complex <input checked="" type="checkbox"/>	Absent <input type="checkbox"/>
Utilitate		Contract de cercetare <input type="checkbox"/>	Cercetare internă <input type="checkbox"/>	Utilare <input checked="" type="checkbox"/>	Altul <i>[se detaliază]</i>
Redactarea lucrării		Insuficient <input type="checkbox"/>	Satisfăcător <input type="checkbox"/>	Bine <input type="checkbox"/>	Foarte bine <input checked="" type="checkbox"/>
Partea grafică, desene		Insuficientă <input type="checkbox"/>	Satisfăcătoare <input type="checkbox"/>	Bună <input type="checkbox"/>	Foarte bună <input checked="" type="checkbox"/>
Realizarea practică	Contribuția autorului	Insuficientă <input type="checkbox"/>	Satisfăcătoare <input type="checkbox"/>	Mare <input type="checkbox"/>	Foarte mare <input checked="" type="checkbox"/>
	Complexitatea temei	Simplă <input type="checkbox"/>	Medie <input type="checkbox"/>	Mare <input type="checkbox"/>	Complexă <input checked="" type="checkbox"/>
	Analiza cerințelor	Insuficient <input type="checkbox"/>	Satisfăcător <input type="checkbox"/>	Bine <input type="checkbox"/>	Foarte bine <input checked="" type="checkbox"/>
	Arhitectura	Simplă <input type="checkbox"/>	Medie <input type="checkbox"/>	Mare <input type="checkbox"/>	Complexă <input checked="" type="checkbox"/>

	Întocmirea specificațiilor funcționale	Insuficientă <input type="checkbox"/>	Satisfăcătoare <input type="checkbox"/>	Bună <input type="checkbox"/>	Foarte bună <input checked="" type="checkbox"/>
	Implementarea	Insuficientă <input type="checkbox"/>	Satisfăcătoare <input type="checkbox"/>	Bună <input type="checkbox"/>	Foarte bună <input checked="" type="checkbox"/>
	Testarea	Insuficientă <input type="checkbox"/>	Satisfăcătoare <input type="checkbox"/>	Bună <input type="checkbox"/>	Foarte bună <input checked="" type="checkbox"/>
	Funcționarea	Da <input checked="" type="checkbox"/>	Parțială <input type="checkbox"/>	Nu <input type="checkbox"/>	
Rezultate experimentale		Experiment propriu <input checked="" type="checkbox"/>		Preluare din bibliografie <input type="checkbox"/>	
Bibliografie		Cărți	Reviste	Articole	Referințe web
Comentarii și observații					

În concluzie, se propune:

ADMITEREA PROIECTULUI <input checked="" type="checkbox"/>	RESPINGEREA PROIECTULUI <input type="checkbox"/>
--	---

Data,

29/06/2022

Semnătura conducătorului științific,



REZUMATUL PROIECTULUI

Scopul lucrării este compararea a minim doi algoritmi de clasificare binară de imagini din punct de vedere al acurateței, preciziei, reapelului, pierderii și specificității clasificării. De asemenea, se va implementa o aplicație cu care pot fi testați algoritmi.

Aplicația conține o parte grafică, utilizată pentru clasificarea propriu-zisă a unor imagini. Imaginile introduse spre clasificare vor aparține unei anumite clase din cele folosite în faza de învățare a modelului, iar aplicația va răspunde, cu un anumit grad de încredere, cărei clase aparține imaginea. De asemenea, aplicația poate fi folosită pentru antrenarea pe noi seturi de date și definirea de noi rețele neuronale convoluționale.

Proiectul a fost implementat utilizând limbajul Python și librării de rețele neuronale specifice acestuia precum tensorflow, pytorch și sklearn.

Lucrarea prezintă în prima parte câteva aspecte teoretice, iar în a doua parte, detalii despre dezvoltarea propriu-zisă a aplicației.

Termenii cheie: python, model, algoritmi de învățare, clasificare binară de imagini

MULȚUMIRI

Aș dori să mulțumesc profesorului coordonator, Prof. Dr. Ing. Liana Stănescu, pentru sprijinul și îndrumarea oferite.

PROLOG

CUPRINSUL

1	INTRODUCERE.....	1
1.1	SCOPUL	1
1.2	MOTIVAȚIA.....	1
2	ETAPA DE ANALIZĂ	3
2.1	CERINȚE FUNCȚIONALE.....	3
2.2	CERINȚE NEFUNCȚIONALE.....	3
2.3	DIAGrame DE UTILIZARE.....	4
2.4	CAZURI DE UTILIZARE	4
2.5	WIREFRAMES.....	6
2.6	TEHNOLOGIILE FOLOSITE ÎN IMPLEMENTAREA APLICAȚIEI	10
3	CÂTEVA ASPECTE TEORETICE UTILIZATE ÎN PROIECT	11
3.1	DESPRE CLASIFICAREA IMAGINILOR	11
3.2	DESPRE ÎNVĂȚAREA AUTOMATĂ	12
3.3	DESPRE REȚELELE NEURONALE.....	14
3.4	CE ESTE UN TENSOR?.....	18
3.5	CNN.....	18
4	DESPRE PROIECT	23
4.1.1	<i>Crearea proiectului</i>	<i>23</i>
4.1.2	<i>Librăriile folosite</i>	<i>24</i>
4.1.2.1	Tensorflow și Keras	24
4.1.2.2	Torch	24
4.1.2.3	Tkinter	24
4.1.3	<i>Metricile calculate</i>	<i>25</i>
4.1.4	<i>Citirea datelor.....</i>	<i>26</i>
4.1.5	<i>Augmentarea Datelor.....</i>	<i>30</i>
4.1.6	<i>Crearea modelului</i>	<i>33</i>
4.1.7	<i>Antrenarea</i>	<i>37</i>
4.1.8	<i>Validarea</i>	<i>40</i>
4.1.9	<i>Salvarea modelului și încărcarea modelului</i>	<i>42</i>
4.1.10	<i>Interfața cu utilizatorul</i>	<i>43</i>
4.1.11	<i>Clase și funcții</i>	<i>45</i>
4.1.12	<i>Testare</i>	<i>51</i>
4.1.13	<i>Parametrii ajustabili.....</i>	<i>53</i>

4.1.14	Prezentarea rezultatelor	54
5	CONCLUZII	63
6	MANUAL DE UTILIZARE	64
7	BIBLIOGRAFIE.....	66
A.	CODUL SURSĂ	66
INDEX.....		ERROR! BOOKMARK NOT DEFINED.

LISTA FIGURILOR

FIGURA 1. DIAGRAMA DE UTILIZARE	4
FIGURA 2 WIREFRAME CLASIFICARE IMAGINE	8
FIGURA 3 WIREFRAME ANTRENARE MODEL	9
FIGURA 4 REȚEA NEURONALĂ CLASICĂ	15
FIGURA 5 REȚEA CU MAI MULTE NIVELE	16
FIGURA 6 EXEMPLU RELU	20
FIGURA 7 EXEMPLU DE MAX-POOLING	21
FIGURA 8 EXEMPLU CNN	22
FIGURA 9 CREAREA UNUI NOU PROIECT ÎN PYCHARM	23
FIGURA 10 MATRICE DE CONFUZIE ÎN CAZUL CLASIFICĂRII BINARE	25
FIGURA 11 STRUCTURA DATELOR DE INTRARE	27
FIGURA 12 EXEMPLU DE AUGMENTARE DE DATE	31
FIGURA 13 ARHITECTURA UTILIZATĂ PENTRU CNN	34
FIGURA 14 DIAGRAMA DE CLASE	45
FIGURA 15 ARHITECTURA CU 3 NIVELE	53
FIGURA 16 PIERDERE PENTRU CNN CU 8 STRATURI	54
FIGURA 17 ACURATEȚE PENTRU CNN CU 8 STRATURI	54
FIGURA 18 PRECIZIE PENTRU CNN CU 8 STRATURI	55
FIGURA 19 REAPEL PENTRU CNN CU 8 STRATURI	55
FIGURA 20 SPECIFICITATE PENTRU CNN CU 8 STRATURI	55
FIGURA 21 PIERDERE PENTRU CNN CU 3 STRATURI	57
FIGURA 22 ACURATEȚE PENTRU CNN CU 3 STRATURI	57
FIGURA 23 PRECIZIE PENTRU CNN CU 3 STRATURI	57
FIGURA 24 REAPEL PENTRU CNN CU 3 STRATURI	58
FIGURA 25 SPECIFICITATE PENTRU CNN CU 3 STRATURI	58
FIGURA 26 PIERDERE SET DE DATE MIC	59
FIGURA 27 ACURATEȚE SET DE DATE MIC	60
FIGURA 28 PRECIZIE SET DE DATE MIC	60
FIGURA 29 REAPEL SET DE DATE MIC	60
FIGURA 30 SPECIFICITATE SET DE DATE MIC	61
FIGURA 31 INTERFAȚA CU UTILIZATORUL	64

LISTA TABELELOR

TABEL 1 COMPARAȚIE KERAS-PYTORCH CNN ANTRENAMENT	56
TABEL 2 COMPARAȚIE KERAS-PYTORCH CNN VALIDARE.....	56
TABEL 3 COMPARAȚIE KERAS-PYTORCH 3 NIVELE ANTRENAMENT	58
TABEL 4 COMPARAȚIE KERAS-PYTORCH 3 NIVELE VALIDARE	59
TABEL 5 COMPARAȚIE KERAS-PYTORCH 8 NIVELE ANTRENAMENT SET DE DATE DE DIMENSIUNE REDUSA.....	61
TABEL 6 COMPARAȚIE KERAS-PYTORCH 8 NIVELE VALIDARE SET DE DATE DE DIMENSIUNE REDUSA	61

1 INTRODUCERE

1.1 Scopul

Această lucrare a fost concepută cu scopul de a oferi o comparație a mai multor algoritmi de clasificare în aplicațiile cu rețele neuronale, implementați utilizând două librării python. Comparația se face pe baza realizării unui studiu experimental asupra unui set de date de dimensiuni mari.

După realizarea experimentului, algoritmii creați vor fi folosiți într-o aplicație ce permite introducerea unei imagini și clasificarea conținutului acestuia.

1.2 Motivația

Motivația alegerii proiectului este faptul că clasificarea automată de imagini a devenit un subiect tot mai răspândit și cu aplicabilitate în diverse domenii.

Un prim domeniu de utilizare este cel auto, odată cu popularizarea mașinilor care se conduc singure. Pentru aceasta este nevoie de monitorizare continuă a mediului înconjurător pentru a distinge elementele din trafic ce fac posibilă deplasarea automată în siguranță. Printre aceste elemente se numără distingerea semnelor de circulație, a liniilor ce delimitează drumul de mers sau a altor elemente din trafic, statice(copaci, sens giratoriu) sau mobile(alte vehicule).

Un alt domeniu de aplicabilitate interesant este cel medical. Un exemplu, ce a fost utilizat și ca date de intrare în experimentul realizat, este clasificarea unor mamografii pentru a distinge între țesut sănătos și țesut afectat de cancer. Alte exemple de detecție sunt cele ale oaselor rupte, tumori sau diverse tipuri de infecții.

Un sector de aplicabilitate pentru clasificarea de imagini este chiar cel online, de exemplu pentru reglementarea conținutului. Se pot determina paginile ce conțin imagini nerecomandate persoanelor aparținând unor anumite categorii de vârstă(imagini cu conținut explicit).

Desigur că în multe din exemplele de aplicabilitate prezentate nu poate fi vorba de încredere totală în algoritmii utilizați și că în majoritatea cazurilor este nevoie de intervenție umană pentru a re-verifica rezultatele. Spre exemplu, în cazul domeniului auto, este obligatoriu ca șoferul să fie atent la drum și gata să preia controlul mașinii în orice moment. În cazul domeniului medical, diagnosticul nu poate fi bazat complet pe clasificarea automată a imaginilor. Cu toate acestea, clasificarea automată a imaginilor are scopul de a facilita și de a oferi suport în procesul de luare a deciziilor.

Astfel, având atât de multe domenii de aplicabilitate, se pune problema despre care dintre algoritmi existenți și care implementare a acestuia este cea mai eficientă.

2 ETAPA DE ANALIZĂ

Prima etapă în dezvoltarea unui proiect constă în analiza cerințelor.

2.1 Cerințe funcționale

- Aplicația trebuie să suporte minim două implementări de algoritmi de machine learning/deep learning.
- Aplicația trebuie să permită selectarea unei imagini de pe dispozitiv.
- Aplicația trebuie să permită selectarea algoritmului ce va fi folosit în clasificarea imaginii.
- Aplicația trebuie să permită clasificarea unei imagini în una din clasele predefinite.
- Aplicația trebuie să permită afișarea rezultatului clasificării.
- Imaginea introdusă pentru a fi clasificată poate aparține unei singure clase din domeniul ales.
- Aplicația trebuie să permită colectarea parametrilor necesari pentru definirea arhitecturii unei noi rețele neuronale convoluționale.
- Aplicația trebuie să permită antrenarea și validarea pe un set de date binar folosind algoritmul și arhitectura introdusă.
- Aplicația trebuie să permită afișarea metricilor performanței după terminarea antrenării și validării.

2.2 Cerințe nefuncționale

- Datele din aplicație trebuie afișate într-o manieră prietenoasă cu utilizatorul.
- Clasificarea trebuie să nu dureze mai mult de 5 secunde.
- Platforma pe care este suportat algoritmul este cel puțin Windows 10.
- Pentru ca aplicația să ruleze este necesar ca pe dispozitiv să fie instalat Python3.
- Aplicația va folosi modele antrenate în prealabil pe anumite seturi de date de dimensiuni mari.

- Antrenarea modelului nu are o limită temporală.

2.3 Diagrame de Utilizare

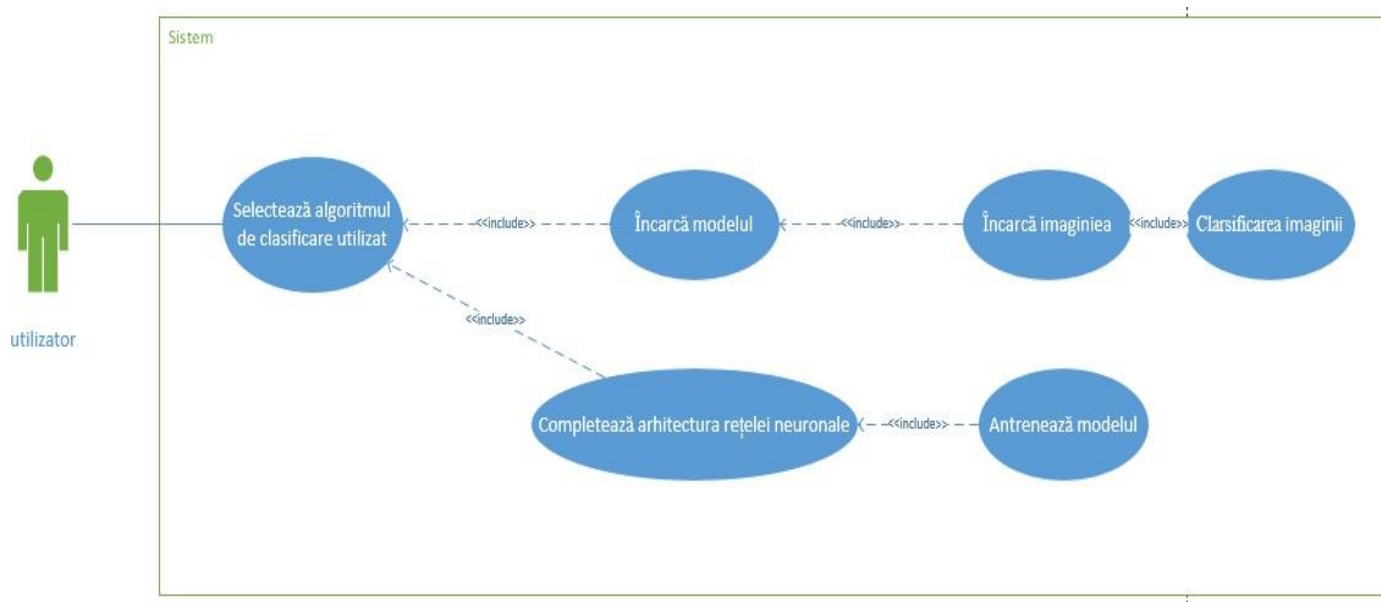


Figura 1. Diagrama de Utilizare

2.4 Cazuri de Utilizare

1. Clasificarea unei mamografii cu țesut cancerigen folosind algoritmul CNN din pytorch
 - Utilizatorul deschide aplicația desktop.
 - Utilizatorul selectează opțiunea de algoritm de clasificare CNN din pytorch.
 - Utilizatorul apasă butonul de selectare a modelului.
 - Utilizatorul selectează de pe dispozitiv modelul pytorch antrenat.
 - Utilizatorul apasă butonul de încărcare de imagine
 - Utilizatorul selectează o imagine reprezentând o mamografie pozitivă de pe dispozitiv
 - Utilizatorul apasă butonul de clasificare.

- Sistemul clasifică imaginea introdusă cu eticheta cancer și o valoare a gradului de încredere.

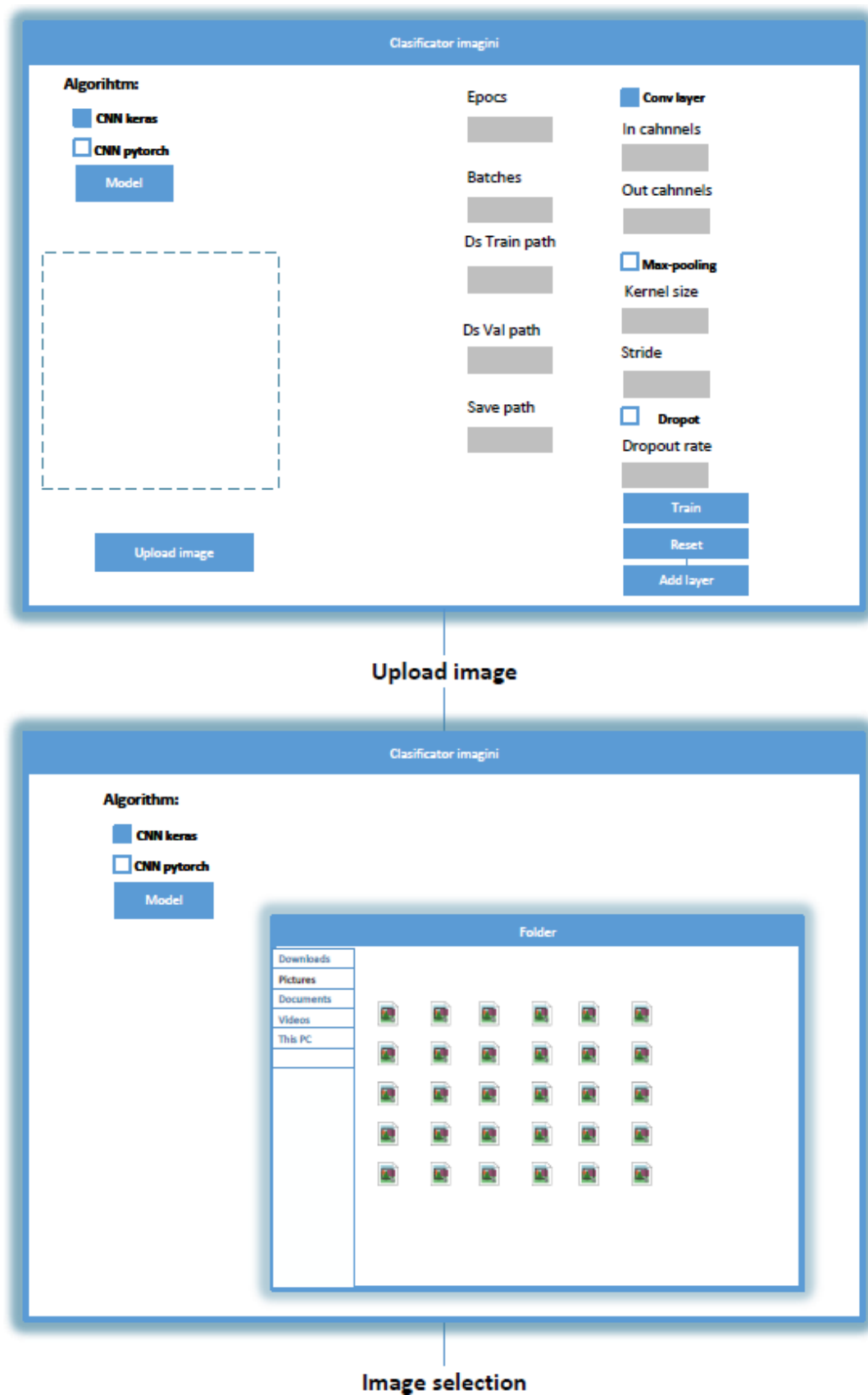
2. Antrenarea unei rețele neuronale convoluționale în pytorch

- Utilizatorul deschide aplicația desktop.
- Utilizatorul selectează opțiunea de algoritm de clasificare CNN din pytorch.
- Utilizatorul completează câmpurile număr de epoci, dimensiunea lotului, calea către seturile de date de antrenament și validare, calea unde va fi salvat modelul antrenat.
- Utilizatorul adaugă nivelele din rețeaua CNN (nivel de convoluție, max-pooling, linear, sigmoid, dropout).
- Utilizatorul apasă butonul antrenează modelul.
- Utilizatorul așteaptă rezultatul antrenamentului.
- Sistemul afișează metricile de performanță calculate după etapele de antrenament și validare.

3. Resetarea modelului

- Utilizatorul deschide aplicația desktop.
- Utilizatorul selectează opțiunea de algoritm de clasificare CNN din pytorch.
- Utilizatorul completează câmpurile număr de epoci, dimensiunea lotului, calea către seturile de date de antrenament și validare, calea unde va fi salvat modelul antrenat.
- Utilizatorul adaugă la rețeaua CNN două straturi de convoluție consecutive.
- Utilizatorul apasă butonul resetare model.
- Utilizatorul reia construcția arhitecturii de la început

2.5 Wireframes



Classify


Clasificator imagini

Algorithm:

☒ CNN keras
 ☐ CNN pytorch

Model

Cnn_Pytorch_model_10layers.pth



Normal 80%

Epochs

Batches

Ds Train path

Ds Val path

Save path

☒ Conv layer

In cahnnels

Out cahnnels

☐ Max-pooling

Kernel size

Stride

☐ Dropot

Dropout rate

Train

Reset

Add layer

Figura 2 Wireframe clasificare imagine

Clasificator imagini

Algorihtm:

☒ CNN keras
 ☐ CNN pytorch

Model

Upload image

Epochs

Batches

Ds Train path

Ds Val path

Save path

☒ Conv layer

In cahnnels

Out cahnnels

☐ Max-pooling

Kernel size

Stride

☐ Dropot

Dropout rate

Train

Reset

Add layer

Completare campuri

Reset

8

Completare campuri Reset

Clasificator imagini

Algorihtm:

☒ CNN keras

☐ CNN pytorch

Model

Upload image

Epocs

Batches

Ds Train path

Ds Val path

Save path

☒ Conv layer

In cahnnels

Out cahnnels

☐ Max-pooling

Kernel size

Stride

☐ Dropot

Dropout rate

Train

Reset

Add layer

Antrenare

Clasificator imagini

Algorihtm:

☒ CNN keras

☐ CNN pytorch

Model

Upload image

Epocs

Batches

Ds Train path

Ds Val path

Save path

Metrics display

☒ Conv layer

In cahnnels

Out cahnnels

☐ Max-pooling

Kernel size

Stride

☐ Dropot

Dropout rate

Train

Reset

Add layer

Figura 3 Wireframe antrenare model

2.6 Tehnologiile folosite în implementarea aplicației

Aplicația a fost implementată utilizând limbajul Python3, iar IDE-ul folosit este PyCharm.

„Python este un limbaj de programare dinamic ce permite atât programarea folosind paradigma orientată pe obiecte cât și cea imperativă, funcțională sau procedurală. Sintaxa este mai puțin strictă față de cea a altor limbaje precum C. De exemplu blocurile de cod precum funcțiile sau instrucțiunile dintr-o instrucțiune de tip for sunt delimitate prin indentare în loc de acolade. De asemenea este permisă tipizarea dinamică sau utilizarea variabilelor fără ca acestea să fie declarate în prealabil și se ocupă de alocarea și dealocarea memoriei automat. ”¹

Python este un limbaj interpretabil ceea ce îl poate face mai lent față de alte limbaje compilabile.

Avantajul în a folosi Python în programe ce utilizează machine learning este existența multor librării ce implementează algoritmi populari în acest tip de aplicații. De asemenea conține multe librării de procesare a datelor precum pandas sau de vizualizare precum matplotlib.

Pycharm este unul dintre cele mai populare IDE-uri folosite pentru a dezvolta aplicații scrise în Python. Permite rularea și depanarea programelor și, de asemenea, evidențiază erorile de sintaxă, afișează sugestii de autocompletare și indexează proiectul pentru a putea naviga rapid la orice simbol. Pycharm are integrat, de asemenea, suport pentru testarea modulară și versionarea fișierelor.

¹ [Python\(programming language\)](#)

3 CÂTEVA ASPECTE TEORETICE UTILIZATE ÎN PROIECT

3.1 Despre clasificarea imaginilor

Pentru a putea înțelege clasificarea imaginilor, putem face o comparație cu regăsirea informației. Regăsirea informației, în viața de zi cu zi, reprezintă un anumit mod, un algoritm, de a găsi informațiile necesare într-un context dat. De exemplu, găsirea unei flori pe un câmp.

În informatică, regăsirea informațiilor are un scop similar și trebuie să vizeze individual mai multe tipuri de date, precum text, imagine, video, audio. Pentru fiecare dintre aceste tipuri de date sunt utilizate abordări multiple cu diferite grade de eficiență și compromisuri. Astfel algoritmi folosiți au crescut în timp în eficiență, dar și în complexitate.

Procesul de regăsire începe atunci când un utilizator introduce o interogare într-un sistem. Un exemplu de interogare este căutarea unui text într-un motor de căutare web. În funcție de aplicație, interogările pot fi documente text, imagini, fișiere audio sau video, dar și fișiere cu alte formate.

Un exemplu de interogare de imagine este încărcarea unei imagini ce conține un raton iar rezultatul așteptat este mai multe imagini reprezentând același animal. În spatele interogării, imaginea încărcată este analizată și sunt extrase anumite caracteristici precum culoare, formă, vârful, dimensiuni. Aceste caracteristici sunt apoi comparate cu caracteristicile altor imagini aflate într-o bază de date și care au fost extrase în prealabil și se calculează cât de asemănătoare sunt. Apoi sunt returnate imaginile ale căror caracteristici sunt cele mai asemănătoare cu caracteristicile imaginii din interogare.

În cazul clasificării de imagini, se dorește asocierea unei imagini cu una sau mai multe etichete. De asemenea, clasificarea poate fi binară în cazul în care există doar două clase posibile (de exemplu plămân afectat de pneumonie sau nu) sau clasificare multi-clasă atunci când există mai mult de două clase posibile (de exemplu lea, floarea-soarelui, margaretă, pădărie). Un exemplu de asociere a unei imagini cu mai multe etichete poate proveni din domeniul medical, în cazul în care în urma unei radiografii se poate deduce că pacientul suferă de mai mult de o boală.

Verificarea și clasificarea manuală a imaginilor ar putea fi o sarcină obositoare, mai ales atunci când acestea sunt în număr mare și, prin urmare, ar fi foarte util dacă am putea automatiza întreg acest proces folosind învățarea automată.

Câteva exemple de clasificare a imaginilor includ:

- Etichetarea unei mamografii drept cancer sau nu (clasificare binară).
- Clasificarea unei cifre scrise de mână (clasificare multclasă).
- Asocierea cu un nume a unei fotografii a unei fețe (clasificare multclasă).

În general, pașii urmați pentru a dezvolta un program de clasificare de imagini sunt următorii:

- Alegerea setului de date - de obicei cu cât este mai amplu, cu atât rezultatele sunt mai bune. Ideal este să existe un echilibru între numărul de imagini din fiecare clasă a setului de date.
- Prelucrarea datelor - Scopul acestui proces este de a îmbunătăți caracteristicile imaginii prin suprimarea distorsiunilor nedorite și astfel încât modelele să poată beneficia de aceste date îmbunătățite atunci când execută operațiile. Un alt avantaj al preprocesării ar fi generarea de noi date de intrare folosind funcții de rotație, scalare, translație sau estompă asupra setului inițial de date și apoi folosirea imaginilor generate în procesul de antrenare.
- Etapa de învățare - algoritmi de deep learning sau machine learning sunt utilizați pentru a identifica diverse pattern-uri ale imaginii, caracteristici care ar putea fi unice pentru o anumită clasă și care, mai târziu, vor ajuta modelul să facă diferența între diferite clase. Acest proces în care modelul învață caracteristicile din setul de date se numește antrenarea modelului.
- Etapa de validare - Acest pas constă în clasificarea imaginilor dintr-un alt set decât cele folosite în etapa de antrenare și apoi compararea rezultatului clasificării cu eticheta reală pentru a determina scorul acurateței, preciziei, reapelului, specificității cu care clasificarea a fost realizată. Setul de date de validare este încă necunoscut modelului. În faza de evaluare, parametrii modelului se mai pot schimba în funcție de rezultatele evaluării.
- Etapa de testare – Acest pas constă în clasificarea imaginilor dintr-un alt set decât cele folosite în etapele de antrenare și validare. Acest set de date trebuie să fie complet necunoscut modelului și prezicerile asupra lui nu vor influența parametrii învățați.

3.2 Despre învățarea automată

Învățarea automată este un tip de inteligență artificială care oferă calculatoarelor capacitatea de a învăța fără a fi programate în mod explicit. Învățarea automată se concentrează pe dezvoltarea de programe de calculator care se pot schimba atunci când sunt expuse la noi date.

Învățarea automată implică antrenarea unui computer folosind un set de date de antrenament și utilizarea acestui antrenament pentru a prezice proprietățile unor date noi. De exemplu, putem antrena un

computer dându-i 1000 de imagini cu ratoni și încă 1000 de imagini care nu sunt ale unui raton și îi putem spune de fiecare dată dacă o imagine este sau nu raton. Apoi, dacă arătam computerului o nouă imagine, atunci computerul ar trebui să poată spune dacă această nouă imagine este sau nu un raton cu un anumit grad de încredere.

Procesul de antrenare și predicție implică utilizarea unor algoritmi specializați. Noi dăm datele de antrenament către un algoritm, iar algoritmul folosește aceste date de antrenament pentru a oferi predicții cu privire la un nou set de date.

Un model este un sistem de întrebări-răspunsuri care se ocupă de procesarea datelor.

Tipuri de algoritmi de învățare automată:

1. Regresie (Predicție)

În general, folosim algoritmi de regresie când dorim să prezicem valori continue.

Algoritmi de regresie:

- Regresie Lineară
- Regresie Polinomială
- Regresie Exponențială
- Regresie Logaritmică

2. Clasificare

Folosim algoritmi de clasificare pentru a prezice un set de date aparținând diferitor categorii.

Algoritmi de clasificare:

- KNN
- SVM
- Naive Bayes

3. Clustering

Folosim algoritmi de clustering pentru sumarizarea și structurarea datelor.

Algoritmi de clustering:

- K-means
- DBSCAN

4. Detectarea anomaliilor

Folosiți pentru detectarea activităților anormale sau cazuri speciale cum ar fi detectarea fraudei.

5. Sisteme de recomandare

3.3 Despre rețelele neuronale

„O rețea neuronală este un proces care încearcă să stabilească relațiile dintre componentele unui set de date, imitând funcționarea creierului uman prin faptul ca are la bază sisteme de neuroni de natură organică sau artificială. Un neuron dintr-o rețea neuronală este o funcție matematică care colectează și clasifică informații în funcție de o arhitectură specifică. O rețea neuronală conține straturi de noduri interconectate. Fiecare nod este cunoscut sub numele de perceptron și este similar cu o regresie liniară multiplă. Perceptronul alimentează semnalul produs de o regresie liniară multiplă într-o funcție de activare care poate fi neliniară.”²

În general, o rețea neuronală conține trei nivele:

- Nivelul de intrare
- Nivelul de procesare – este ascuns și conține noduri interconectate, proiectate să fie similare cu neuronii și sinapsele din creierul organismelor umane.
- Nivelul de ieșire

De cele mai multe ori, rețelele neuronale se adaptează la nivelul de intrare fără a afecta nivelul de ieșire schimbându-și structura în funcție de informațiile care circulă prin rețea.

² [Neural Network](#)

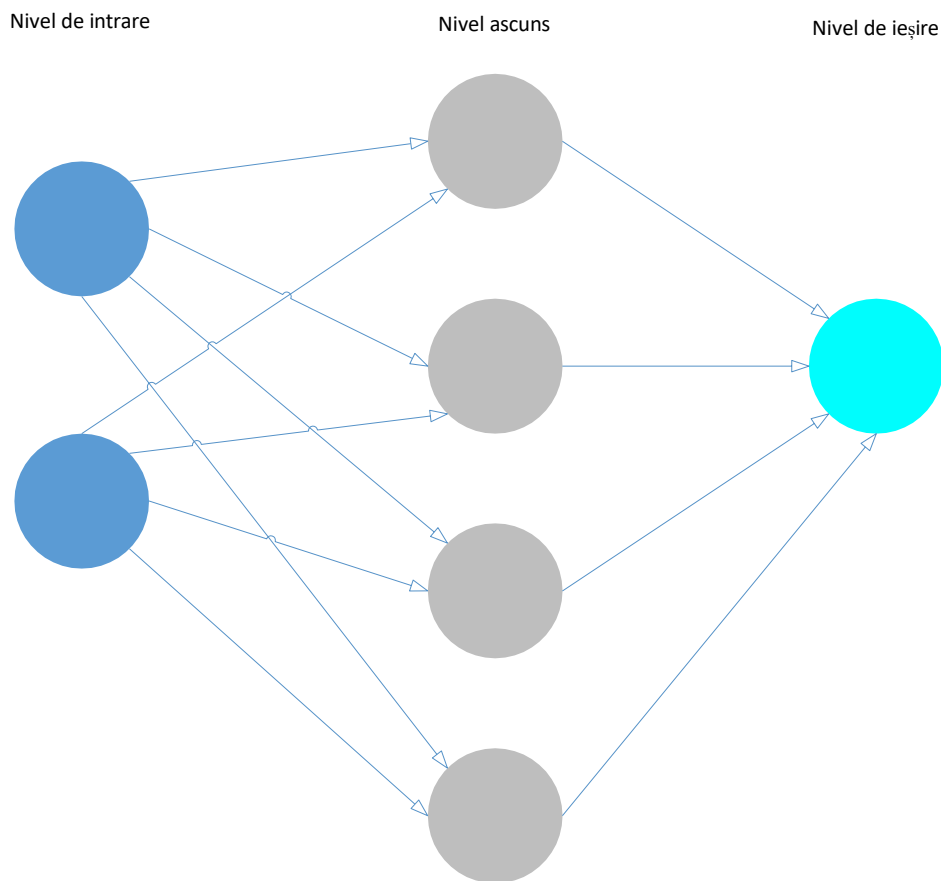


Figura 4 Rețea neuronală clasică

Diferența dintre rețelele neuronale și deep learning constă în numărul de nivele din rețea. O rețea neuronală care conține mai mult de trei nivele este considerată algoritm de deep learning.

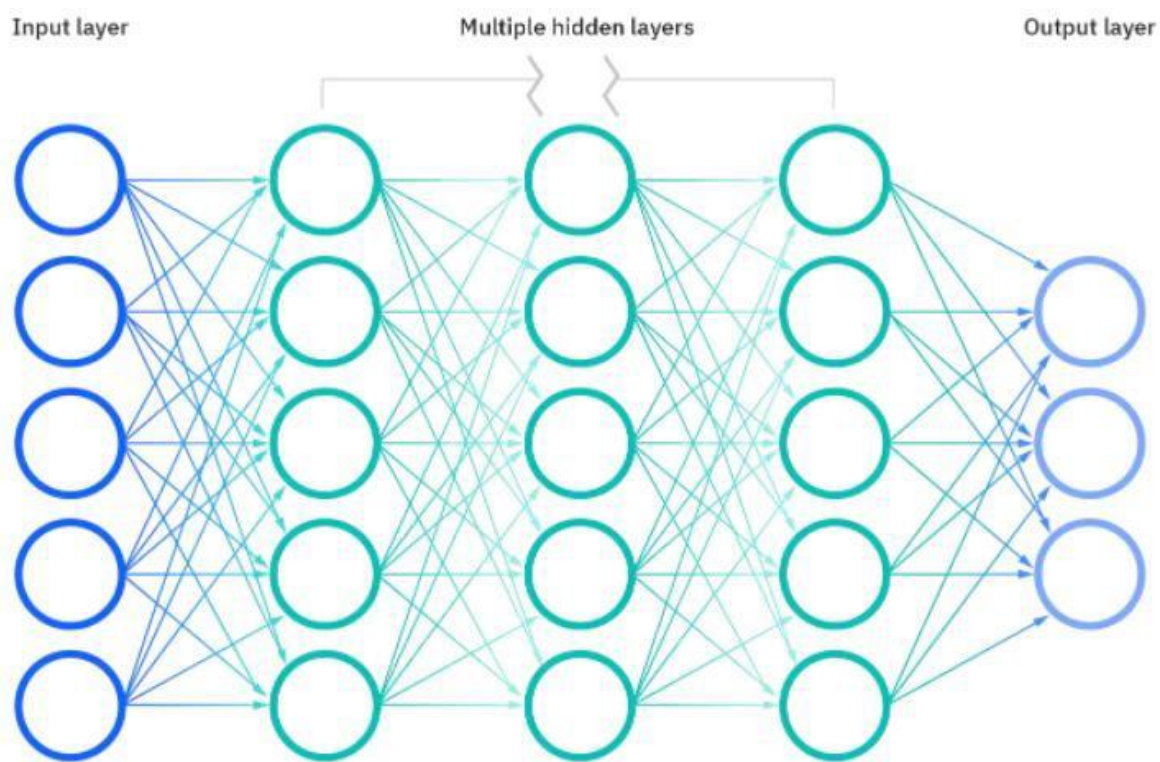


Figura 5 Rețea cu mai multe nivele³

„Neuronul artificial primește una sau mai multe intrări care au o pondere ce se ajustează odată ce învățarea avansează. Fiecare neuron procesează semnalele de intrare și apoi trimite rezultatul către următorul neuron la care este conectat. Rezultatul procesării este un număr real calculat pe baza unei funcții, de obicei nelineară, prin care este trecută suma ponderilor intrărilor. Această funcție este cunoscută sub denumirea de funcție de activare sau funcție de transfer. Funcțiile de transfer au de obicei o formă sigmoidă, dar pot lua și alte forme. Ele sunt, de asemenea, adesea monoton crescătoare, continue, diferențiabile și mărginite.”⁴

Neuronii pot avea un prag astfel încât un semnal este trimis numai dacă semnalul agregat depășește acel prag.

O funcție de transfer poate avea formula:

³ [Neural-Networks](#)

⁴ [Neuron Artificial](#)

$$\sum w_i x_i + \text{bias} = w_1 x_1 + w_2 x_2 + w_3 x_3 + \text{bias}$$

$$\text{output} = f(x) = 1 \text{ dacă } \sum w_1 x_1 + b \geq 0; 0 \text{ dacă } \sum w_1 x_1 + b < 0''^5$$

„Odată ce un strat de intrare este determinat, ponderile sunt atribuite. Toate intrările sunt apoi înmulțite cu ponderile lor respective și apoi însumate. Ulterior, ieșirea este trecută printr-o funcție de activare, care determină ieșirea. Dacă acea ieșire depășește un anumit prag, activează nodul, pasând datele următorului strat din rețea. Aceasta are ca rezultat ieșirea unui nod să devină intrarea următorului nod. Acest proces de transmitere a datelor de la un strat la stratul următor definește rețeaua neuronală ca o rețea feedforward.”⁵

De obicei, neuronii sunt agregați în straturi. Straturi diferite pot efectua transformări diferite asupra intrărilor lor. Semnalele călătoresc de la primul strat (stratul de intrare), la ultimul strat (stratul de ieșire), eventual după ce au traversat straturile de mai multe ori.

„Învățarea propriu-zisă se face prin compararea rezultatului din nivelul de ieșire, după ce o intrare din setul de date de antrenament a fost procesată, și rezultatul real, iar rezultatul comparației este denumit eroare. Rețeaua neuronală ajustează apoi ponderile la fiecare procesare pe baza unei reguli de învățare care ține cont de eroare. Odată cu înaintarea procesului de învățare, nivelul de ieșire conține rezultate tot mai apropiate de rezultatul real, eroarea scade, iar după un număr suficient de mare de procesări se poate încheia faza de învățare.”⁶

„Un alt termen important în rețele neuronale este funcția de cost care este legată de eliminarea deducerilor incorecte. O funcție de cost folosită în mod obișnuit este eroarea pătratică medie (mean-squared root), care încearcă să minimizeze eroarea pătratică medie dintre ieșirea rețelei și ieșirea dorită.

$$\text{„Cost Function} = MSE = 1/2m \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2''^7$$

„Scopul învățării este de a minimiza funcția de cost pentru a asigura corectitudinea potrivirii pentru orice intrare dată. Pe măsură ce modelul își ajustează ponderile, acesta utilizează funcția de cost pentru

⁵ [Neural Networks](#)

⁶ [Neural Networks](#)

⁷ [Neural Networks](#)

a atinge punctul de convergență sau minimul local. Cu fiecare exemplu de antrenament, parametrii modelului se ajustează pentru a converge treptat la minim.”⁸

3.4 Ce este un tensor?

„În matematică, un tensor este un obiect algebric care descrie o relație multiliniară între seturi de obiecte algebrice legate de un spațiu vectorial. Obiectele între care tensorii pot mapa includ vectori și scalari și chiar alți tensori. Există multe tipuri de tensori, inclusiv scalari și vectori (care sunt cei mai simpli tensori), vectori duali, hărți multilinare între spațiile vectoriale și chiar unele operații, cum ar fi produsul punctual.”⁹

Tensorii sunt pur și simplu obiecte matematice care pot fi folosite pentru a descrie proprietăți fizice, la fel ca scalarii și vectorii. De fapt, tensorii sunt doar o generalizare a scalarilor și a vectorilor, astfel că un scalar este un tensor de rang zero, iar un vector este un tensor de rangul unu.

„Rangul (sau ordinea) unui tensor este definit de numărul de direcții (și, prin urmare, de dimensiunile matricei) necesare pentru a-l descrie. De exemplu, proprietățile care necesită o direcție (primul rang) pot fi descrise complet de un vector coloană 3×1 , iar proprietățile care necesită două direcții (tensori de rangul doi), pot fi descrise prin 9 numere, ca o matrice 3×3 . Ca atare, în general, un tensor de rang al n -lea poate fi descris prin 3^n coeficienți.”¹⁰

La fel ca vectorii și matricele, tensorii pot fi reprezentați în Python folosind matricea N -dimensională (`ndarray`). Un tensor poate fi definit cu constructorul `array()` ca o listă de liste. Ca și în cazul matricelor, putem efectua operații aritmetice între tensori.

Am folosit în proiect tensorii pentru a descrie setul de date. O imagine din setul de date este de fapt un tensor de rang 3, iar rangurile tensorului reprezintă, de fapt, lungimea, înălțimea și numărul de canale al imaginii.

3.5 CNN

⁸ [Neural Networks](#)

⁹ [Tensor](#)

¹⁰ [What is tensor?](#)

CNN vine de la Convolutional Neural Network și este un tip de rețea neuronală adaptat pentru analiza și identificarea datelor vizuale. Mai exact, modelul CNN este specializat în clasificarea imaginilor sau recunoașterea imaginilor sau a modelelor din imagini.

Modelul are la bază concepte din algebra liniară precum înmulțirea matricelor pe care le utilizează pentru a recunoaște modelele din imagini.

„În loc să preproceseze datele pentru a obține caracteristici precum texturi și forme, un model de tip CNN preia doar datele brute de pixeli ale imaginii ca intrare și învață cum să extragă aceste caracteristici și, în cele din urmă, deduce ce obiect constituie ele.”¹¹

CNN primește o hartă a caracteristicilor de intrare: o matrice tridimensională în care primele două dimensiuni corespund lungimii și lățimii imaginilor în pixeli. Dimensiunea celei de-a treia dimensiuni este 3 (corespunzând celor 3 canale ale unei imagini color: roșu, verde și albastru).

Convoluția extrage părți ale hărții de caracteristici de intrare și aplică filtre pentru a calcula caracteristici noi, producând o hartă de caracteristici de ieșire sau o caracteristică convolută (care poate avea o dimensiune și o adâncime diferite decât harta caracteristică de intrare). Convoluțiile sunt definite de doi parametri:

- Dimensiunea bucaților din imagine care sunt extrase adică dimensiunea filtrelor (de obicei 3x3 sau 5x5 pixeli).
- Adâncimea hărții caracteristicilor de ieșire, care corespunde numărului de filtre care sunt aplicate.

În timpul convoluției, filtrele parcurg harta caracteristicilor de intrare orizontal și vertical, câte un pixel o dată.

Pentru fiecare pereche filtru-bucată de matrice, CNN efectuează înmulțirea de elemente ale matricei de filtru și a matricei de bucăți din imagine, apoi însumează toate elementele matricei rezultate pentru a obține o singură valoare. Fiecare dintre aceste valori rezultate pentru fiecare pereche este apoi scoasă în matricea de caracteristici convolute. În timpul antrenamentului, CNN învață valorile optime pentru matricele de filtre, care îi permit să extragă caracteristici semnificative (texturi, margini, forme) din harta caracteristicilor de intrare. Pe măsură ce numărul de filtre crește, la fel crește și numărul de caracteristici pe care CNN le poate extrage. Timpul de antrenament crește și el pe măsură ce se adaugă

¹¹ [convolutional-neural-networks](#)

mai multe filtre. În plus, fiecare filtru adăugat în rețea oferă o valoare incrementală mai mică decât cel anterior.

După fiecare operație de convoluție, CNN aplică o transformare Rectified Linear Unit (ReLU) caracteristicii convolutive, pentru a introduce neliniaritatea în model. Funcția ReLU returnează x pentru toate valorile lui $x > 0$ și returnează 0 pentru toate valorile lui $x \leq 0$.

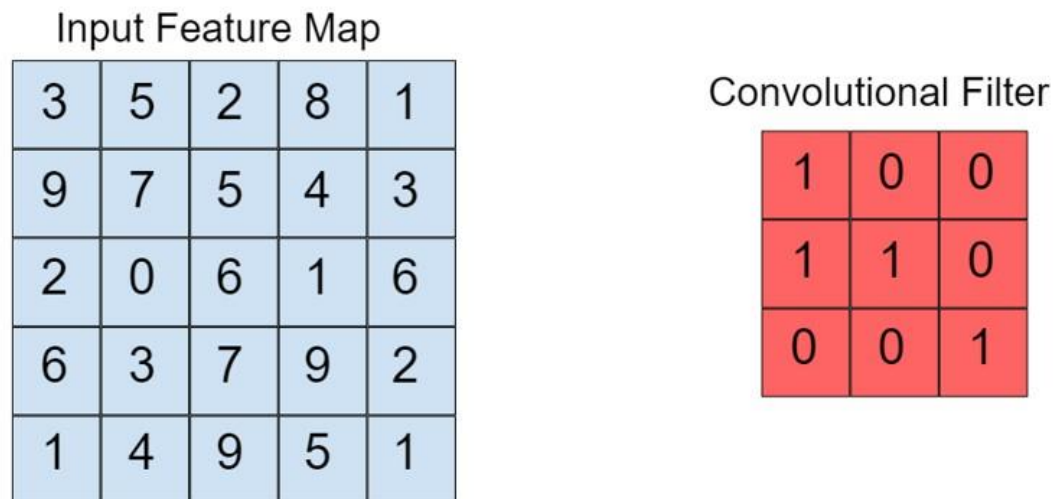


Figure 4a. **Left:** A 5x5 input feature map (depth 1). **Right:** a 3x3 convolution (depth 1).

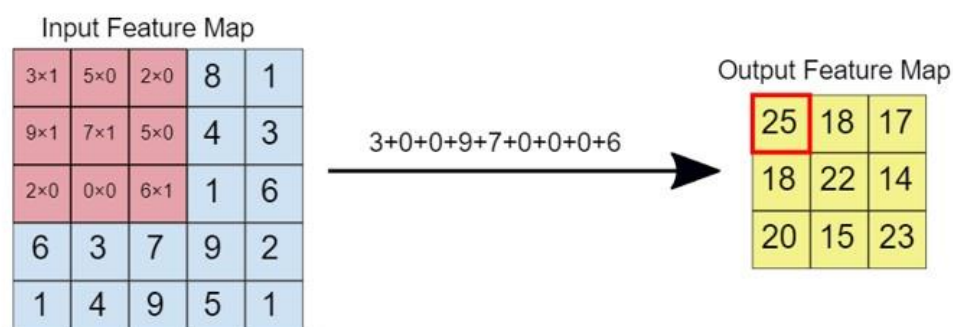


Figura 6 exemplu ReLU ¹²

După ReLU vine o etapă de pooling, în care CNN eșantionează caracteristica convolută (pentru a economisi timpul de procesare), reducând numărul de dimensiuni ale hărții caracteristicilor, păstrând în același timp cele mai critice informații despre caracteristică. Un algoritm comun utilizat pentru acest proces se numește max pooling.

¹² [convolutional-neural-networks](#)

Max pooling funcționează într-un mod similar cu convoluția. Glisăm peste harta caracteristicilor și extragem bucăți de o dimensiune specificată. Pentru fiecare bucată, valoarea maximă este calculată pe o nouă hartă a caracteristicilor și toate celelalte valori sunt eliminate. Operațiile de max-pooling au doi parametri:

- Dimensiunea filtrului max-pooling (de obicei 2x2)
- Distanța, în pixeli, care separă fiecare bucată extrasă. Spre deosebire de convoluție, în care filtrele se deplasează peste harta caracteristică pixel cu pixel, în max-pooling, pasul determină locațiile în care este extrasă fiecare bucată. Pentru un filtru 2x2, un pas de 2 specifică faptul că operația de max-pooling va extrage toate bucățile 2x2 care nu se suprapun din harta caracteristicilor.

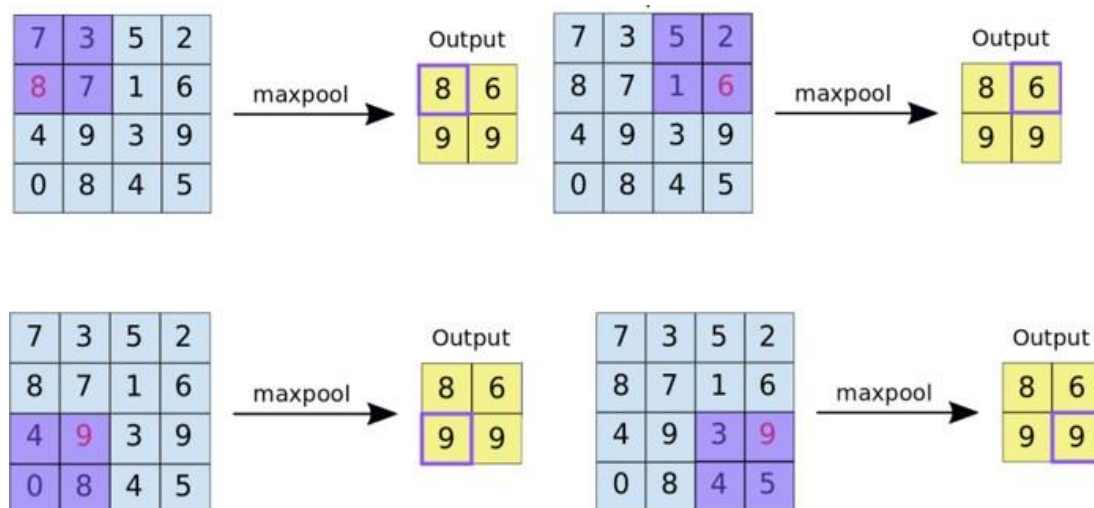


Figura 7 Exemplu de max-pooling¹³

La sfârșitul unei algoritmi CNN se formează unul sau mai multe straturi complet conectate (fiecare nod dintr-un strat este conectat la fiecare nod din alt strat). Sarcina lor este de a efectua clasificarea pe baza caracteristicilor extrase de convoluții. De obicei, stratul final complet conectat conține o funcție de activare softmax, care emite o valoare de probabilitate de la 0 la 1 pentru fiecare dintre etichetele de clasificare pe care modelul încearcă să le prezică.

¹³ [convolutional-neural-networks](#)

În cazul clasificării binare se poate folosi ca strat final complet conectat o funcție de activare sigmoidă, al cărui rezultat va fi o singură valoare și anume o valoare de probabilitate de la 0 la 1 pentru una dintre clase. Implicit, probabilitatea celeilalte clase va fi calculată ca fiind $1 - \text{rezultatul funcției sigmoide}$.

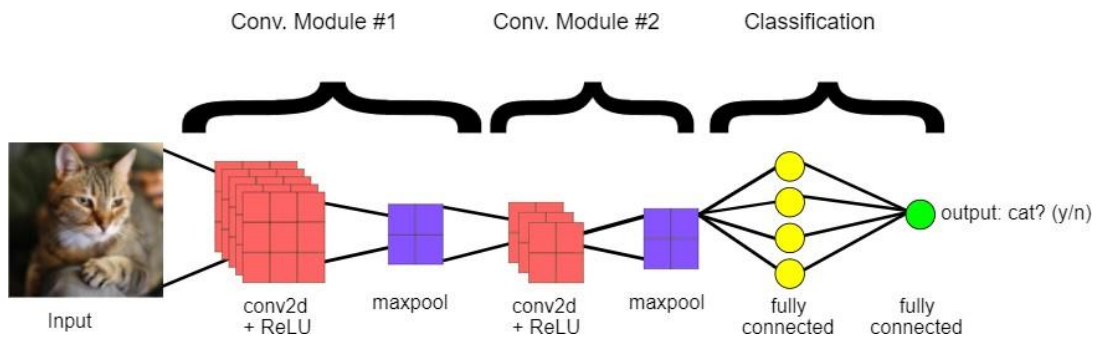


Figura 8 Exemplu CNN¹⁴

¹⁴ [convolutional-neural-networks](#)

4 DESPRE PROIECT

4.1.1 Crearea proiectului

Pentru a crea un nou proiect în Python este necesar să apăsăm pe meniul din stânga sus cu denumirea File->New Project. Apoi va apărea meniul din Figura 13 unde este necesar să completăm numele și locația proiectului și, de asemenea, interpretorul, care în cazul de față este Python3.

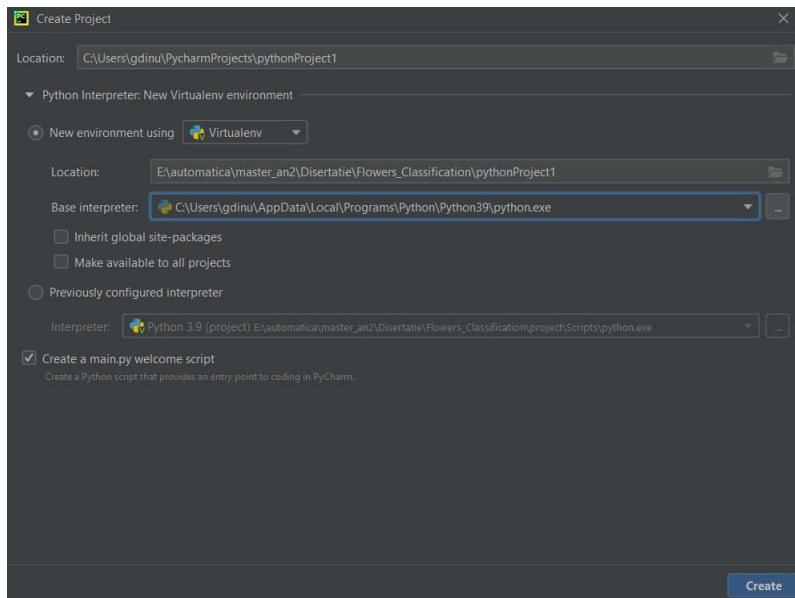


Figura 9 Crearea unui nou proiect în PyCharm

Pentru a crea un fișier nou, este necesar să apăsăm pe meniul din stânga sus cu denumirea File->New, iar apoi va apărea un pop-up pentru selectarea tipului de fișier creat. De aici trebuie să selectăm opțiunea Python File și să introducem numele fișierului.

Pentru rularea proiectului, este necesar să apăsăm pe meniul de sus Run->Run, după care va apărea un pop-up de unde selectăm fișierul ce urmează să fie interpretat.

4.1.2 Librăriile folosite

4.1.2.1 Tensorflow și Keras

TensorFlow este o librărie open-source folosită pentru calcule numerice rapide, adesea folosită pentru implementarea de aplicații de învățare automată sau rețele neuronale. Este creată și menținută de Google. API-urile sunt disponibile pentru limbajul Python, dar există și pentru C++, Java sau Javascript.

Keras este o librărie open-source folosită în aplicații cu rețele neuronale artificiale și funcționează ca o interfață pentru librăria TensorFlow. Keras conține numeroase implementări ale componentelor unei rețele neuronale obișnuite, cum ar fi nivele, funcții de activare, optimizatori și o serie de instrumente pentru a simplifica lucrul cu date de imagine și text.

4.1.2.2 Torch

Torch este o librărie open-source de învățare automată bazat pe limbajul de programare Lua. Pachetul principal din Torch este torch care oferă tensori, care acceptă operații de bază precum indexare, transpunere, redimensionare. Tensorii sunt folosiți de majoritatea celorlalte pachete și formează astfel obiectul de bază al bibliotecii. Tensorul acceptă, de asemenea, operații matematice precum maxim, minim, sumă, distribuții precum cea uniformă, normală și operații BLAS, cum ar fi produsul cartezian, înmulțirea matrice-vector și înmulțirea matrice-matrice.

Pachetul nn este folosit pentru construirea de rețele neuronale. Este împărțit în obiecte modulare care au o interfață comună. Modulele au o metodă forward() și backward() care le permite să se propage înainte și, respectiv, înapoi. Modulele pot fi unite folosind module compozite, cum ar fi Sequential, Parallel și Concat pentru a crea grafice complexe adaptate sarcinilor.

Funcțiile de pierdere (loss function) sunt implementate ca subclase ale clasei Criterion, care are o interfață similară cu clasa Module. Are, de asemenea, metode forward() și backward() pentru calcularea pierderii și respectiv a gradientilor de propagare inversă.

4.1.2.3 Tkinter

Tkinter vine de la Tk interface și este o legătură între Python și setul de instrumente Tk GUI. Tkinter nu este o singură librărie, ci mai degrabă constă din câteva module distincte, fiecare cu funcționalitate separată și propria documentație oficială.

Am folosit aceasta librărie pentru implementarea interfeței cu utilizatorul.

4.1.3 Metricele calculate

Înainte de a putea vorbi despre metricile alese și de a le da o definiție, este necesar să vorbim despre matricea de confuzie.

Matricele de confuzie conțin numărul de valori prezise în raport cu valorile reale. În cazul studiului prezentat, în care clasificarea făcută este binară, elementele matricei de confuzie sunt următoarele:

- TP – true pozitiv, ceea ce înseamnă că valoarea prezisă de clasificator este 1, iar aceasta coincide cu eticheta clasei reale din care face parte imaginea clasificată.
- TN – true negativ, ceea ce înseamnă că valoarea prezisă de clasificator este 1, dar eticheta clasei reale din care face parte imaginea clasificată era 0.
- FP – fals pozitiv, ceea ce înseamnă că valoarea prezisă de clasificator este 0, iar aceasta coincide cu eticheta clasei reale din care face parte imaginea clasificată.
- FN – fals negativ, ceea ce înseamnă că valoarea prezisă de clasificator este 0, dar eticheta clasei reale din care face parte imaginea clasificată era 1.

Astfel, forma matricei de confuzie în cazul clasificării binare are forma:

Predicted Class	
True Class	True Positive (TP)
	False Negative (FN)
True Class	False Positive (FP)
	True Negative (TN)

Figura 10 Matrice de confuzie în cazul clasificării binare

Pe baza elementelor din matricea de confuzie se pot defini metricile de care ținem cont în acest studiu experimental ca fiind:

1. Acuratețe – reprezintă raportul dintre valorile prezise corect de clasificator și numărul total de elemente prezise. Astfel acuratețea poate fi extrasă din matricea de confuzie ca fiind

$$(TP+TN) / (TP+FP+TN+FN)$$
2. Precizia – reprezintă raportul dintre valorile pozitive prezise corect de clasificator și numărul total de valori pozitive prezise. Astfel precizia poate fi extrasă din matricea de confuzie ca fiind

$$(TP) / (TP+FP)$$
3. Reapel sau senzitivitate – reprezintă raportul dintre valorile de 1 prezise corect de clasificator și numărul total de elemente aparținând clasei 1 în realitate. Astfel reapelul poate fi extras din matricea de confuzie ca fiind

$$(TP) / (TP+FN)$$
4. Specificitatea – reprezintă raportul dintre valorile de 0 prezise corect de clasificator și numărul total de elemente aparținând clasei 0 în realitate. Astfel, specificitatea poate fi extrasă din matricea de confuzie ca fiind

$$(TN) / (TN+FP)$$

4.1.4 Citirea datelor

Imaginile folosite în antrenament și validare sunt stocate în memoria internă. De obicei, ele sunt împărțite deja în una din clasele ce pot rezulta în urma clasificării. Astfel structura de organizare a imaginilor în directoare cuprinde câte un director pentru fiecare clasă, iar în interiorul directorului se regăsesc doar imaginile aparținând clasei respective.

Un exemplu de structură de organizare pentru un set de date cu două clase ce cuprinde imagini cu țesut normal sau țesut cancerigen este următorul:

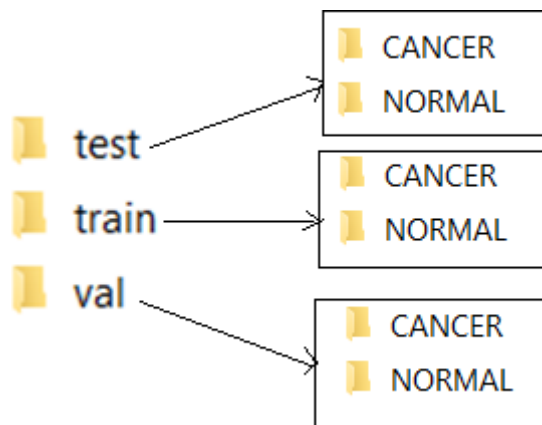


Figura 11 Structura datelor de intrare

Fiecare dintre directoarele test, train și val conțin directoarele normal și cancer. Imaginile din directorul train sunt folosite pentru învățare, iar cele din directorul val sunt folosite pentru validare. În schimb, imaginile din directorul test nu vor fi folosite direct de program, ci pot fi introduse de utilizator în interfața grafică pentru a face o predicție.

Vom folosi următorii termeni pentru a descrie seturile de date:

- Set de date de antrenament: setul de date pe care modelul este antrenat. Acesta este singurul set de date conform căruia se actualizează ponderile în timpul propagării inverse.
- Set de date de validare: setul de date pe care dorim ca modelul să aibă rezultate bune. În timpul procesului de învățare, adaptăm hiperparametrii astfel încât modelul să facă predicții cât mai apropiate de adevăr, dar pentru aceasta nu utilizăm setul de date de validare. Acesta este utilizat doar pentru a vedea performanța modelului și apoi pentru a decide cum putem schimba hiperparametrii pentru ca modelul să poată lucra pe date încă necunoscute lui.
- Set de date de test: este folosit doar pentru calculul performanței. La fel ca în cazul setului de date de validare, nu este folosit propriu-zis în etapa de antrenare. Diferența dintre setul de date de validare și setul de date de test este că, în cazul setului de date de test acesta nu este folosit nici măcar pentru a rafina ponderile, ci îl folosim doar pentru a vedea cât de bine este capabil modelul nostru să generalizeze. În schimb, setul de date de validare va deveni cunoscut modelului prin repetarea adaptării parametrilor.

Setul de date folosit provine de pe platforma Kaggle. „Kaggle, o subsidiară a Google LLC, este o comunitate online de oameni de știință și adepți ai învățării automate. Kaggle permite utilizatorilor să

găsească și să publice seturi de date, să exploreze și să construiască modele într-un mediu de știință a datelor bazat pe web, să colaboreze cu alți oameni de știință și ingineri de învățare automată și să participe la concursuri.”¹⁵

Am realizat experimente pe următorul set de date:

- Imagini de histopatologie mamară:
 - o [breast-histopathology-images](#)

Dar setul de date nu vine deja organizat conform structurii de directoare din figura 11. În schimb, datele sunt organizate în 279 de directoare, fiecare director corespunzând unui pacient. Apoi, pentru fiecare pacient există încă două directoare, primul conținând doar imagini cu celule normale, iar al doilea imagini ce conțin țesut cancerigen. De fapt, fiecare imagine din directorul unui pacient este o secțiune dintr-o mamografie care a fost mărită de 40 de ori. Fiecare imagine este de forma id pacient + coordonata x din imaginea originală de unde s-a efectuat mărirea + coordonata y din imaginea originală de unde s-a efectuat mărirea + clasa în care se încadrează imaginea (cancer sau nu).

Deoarece setul de date este organizat sub această formă, am implementat un program pentru a-l restructura.

Primul pas este citirea căilor tuturor datelor existente. Acestea sunt apoi amestecate. Vom împărți setul de date astfel încât 80% din acestea să fie folosite pentru antrenament, 10% pentru validare și încă 10% pentru testare. După acest pas, vom itera prin toate imaginile și vom parsa numele clasei de care aceasta aparține. Apoi 80% din imagini vor fi mutate în directorul train în subdirectorul clasei respective, 10% din imagini vor fi mutate în directorul val în subdirectorul clasei respective și restul de 10% din imagini vor fi mutate în directorul test în subdirectorul clasei respective.

După acest proces, am ajuns la organizarea din figura 11.

Citirea propriu zisă este diferită în cazul utilizării keras față de cel al utilizării pytorch și voi descrie ambele procese separat mai jos.

În cazul keras, pentru a citi datele, am utilizat funcția `image_dataset_from_directory` din librăria keras. Funcția va returna un set ce conține loturi de imagini, dar care au și etichete asociate cu numele subdirectorului în care se află. Această etichetare este aleasă prin parametrul de intrare al funcției numit

¹⁵ [Kaggle](#)

label și căruia în cod îi dam valoarea inferred. Alți parametri de intrare pentru funcția `image_dataset_from_directory` pe care i-am folosit în cod sunt

- `directory` a cărei valoare este calea directorului principal
- `image_size` pentru care am ales valoarea 50, aceasta fiind și dimensiunea originală a imaginii, ceea ce înseamnă că imaginile vor fi redimensionate la rezoluția de 50x50 indiferent de dimensiunea inițială
- parametrul `shuffle` căruia i-am asignat valoarea `True`, ceea ce înseamnă ca imaginile vor fi amestecate
- parametrul `batch_size` de 64, ceea ce înseamnă ca datele vor fi grupate în seturi de 60 de imagini.

Pentru optimizare, am utilizat asupra setului de date citit funcțiile cache ceea ce înseamnă că prima dată când se va face o iterație asupra datelor acestea vor fi stocate în memoria cache care este mai rapidă decât memoria obișnuită, și funcția `prefetch` care face posibilă preprocesarea datelor și execuția modelului în timpul fazei de antrenament.

În cazul `pytorch`, primul pas a fost utilizarea funcției `ImageFolder` din librăria `torchvision`. Aceasta are ca parametru calea către fișierul rădăcină unde se află datele și încă un parametru numit `transform` care va conține transformările utilizare în augmentarea datelor.

Apoi, folosind funcția `DataLoader` pentru a încărca imaginile și a le transforma în tensori. Ca parametrii avem:

- `dataset`: calea către setul de date ce va fi încărcat
- parametrul `shuffle` căruia i-am asignat valoarea `True`, ceea ce înseamnă că imaginile vor fi amestecate
- `batch_size` cu valoarea de 64, ceea ce înseamnă ca datele vor fi grupate în seturi de 60 de imagini.
- `pin_memory` cu valoarea `False` ceea ce înseamnă că tensorii nu sunt copiați în memoria `cuda` înainte de a fi accesați. În cazul dispozitivului folosit, `cuda` nu este utilizabil.

4.1.5 Augmentarea Datelor

În cazul în care utilizăm un algoritm ce are la baza CNN, la fel ca în cazul altor algoritmi de învățare automată, există riscul de „overfitting”. Mai concret, există posibilitatea ca modelul să devină obișnuit doar cu datele analizate în timpul antrenamentului, dar când este pus în fața situației de a clasifica o nouă imagine, nu este capabil să generalizeze cunoștințele curente asupra noilor date. În cazul în care avem situația de overfitting, valoarea metricei acuratețe din timpul etapei de antrenament este mult mai mare decât valoarea metricei acuratețe din timpul etapei de validare. Același lucru se aplică și pentru restul metricilor calculate și anume precizie, reapele și specificitate.

Două dintre cele mai folosite tehnici pentru a minimiza șansele ca această situație să apară sunt:

1. Augmentarea datelor
2. „Dropout regularization”

În cazul augmentării datelor, procesul constă în multiplicarea datelor de intrare pentru antrenament. Acest lucru se realizează prin crearea artificială de noi imagini din imaginile deja existente. Câteva metode prin care se măresc datele sunt scalarea, rotația, translația, forfecarea.

Din nou, metodele aplicate diferă în cazul folosirii keras față de cazul folosirii pytorch.

Pentru augmentarea datelor, am folosit nivelele de preprocesare din librăria keras și anume:

- RandomFlip pentru a răsturna imaginile atât pe axa verticală cât și pe cea orizontală (prin parametrul mode căruia i-am dat valoarea horizontal_and_vertical)
- RandomRotation pentru a roti imaginea. Aici am folosit parametrul factor cu valoarea de 0.12. Acest parametru reprezintă gradul minim și maxim cu care se rotește imaginea. Deoarece are o valoare pozitivă, se va roti în sensul invers acelor de ceasornic cu o valoare aleatorie în intervalul $[-12\% * 2\pi, 12\% * 2\pi]$
- RandomZoom pentru a mari sau micșora imaginea. Am folosit ca parametru height_factor cu valoarea de 0.2 ceea ce înseamnă o mărire a imaginii cu 20%.

Codul pentru augmentare este :

```
self.data_augmentation = tf.keras.Sequential([
layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical",
input_shape=(self.IMAGE_SIZE,
```

```

self.IMAGE_SIZE,
    layers.experimental.preprocessing.RandomRotation(0.1),
    layers.experimental.preprocessing.RandomZoom(0.1),
]
)

```

De exemplu, pentru o imagine din setul de date utilizat și anume cel cu mamografii, după augmentare, imaginea se va transforma ca în figura 15:

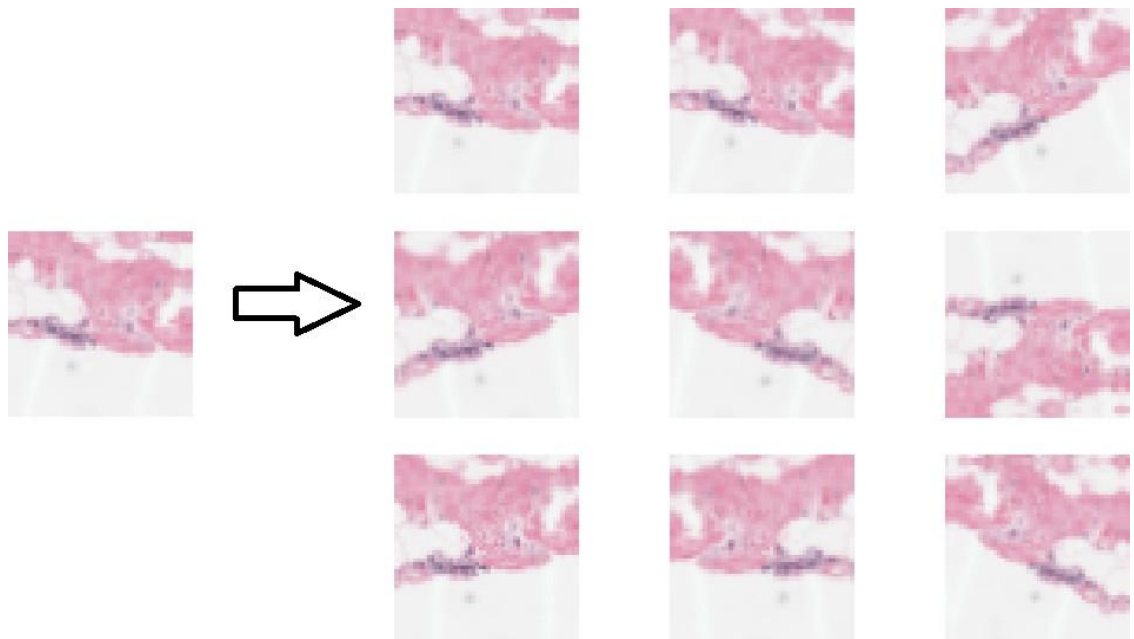


Figura 12 Exemplu de augmentare de date

În cazul pytorch, augmentarea datelor se face aplicând paramentul transform la funcția ImageFolder, parametru ce conține transformările utilizare în augmentarea datelor.

În cazul meu, parametrul este o compoziție de mai mult transformări și anume:

- Resize care primește ca parametru dimensiunea la care datele vor fi redimensionate, în cazul de față 50x50px.
- RandomHorizontalFlip, pentru a răsturna imaginile pe axa orizontală
- RandomVerticalFlip, pentru a răsturna imaginile pe axa verticală

- RandomRotation pentru a roti imaginea. Aici am folosit parametrul degrees cu valoarea de 0.12. Acest parametru reprezintă gradul minim și maxim cu care se rotește imaginea. Fiecare imagine se va roti cu o valoare aleatorie în intervalul [-12, 12] grade. Parametrul fill are valoarea 0, ceea ce înseamnă ca după rotație, golurile vor fi umplute cu valoarea 0.
- ToTensor() – transformă imaginea într-un tensor.
- Normalize, are ca parametru mean=[0.4914, 0.4822, 0.4465] și deviația standard std=[0.2023, 0.1994, 0.2010]. Această transformare normalizează imaginile pentru fiecare canal utilizat. Imaginile inițiale au 3 canale corespunzătoare codului rgb. Transformarea pentru fiecare canal se face după formula:

$$\text{output[channel]} = (\text{input[channel]} - \text{mean[channel]}) / \text{std[channel]}$$

Codul pentru augmentare este:

```
train_transformation = transforms.Compose([
    transforms.Resize((self.IMAGE_SIZE, self.IMAGE_SIZE)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.RandomRotation(0.12),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.4914, 0.4822, 0.4465], std=[0.2023,
0.1994, 0.2010])
])
test_transformation = transforms.Compose([
    transforms.Resize((self.IMAGE_SIZE, self.IMAGE_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.4914, 0.4822, 0.4465], std=[0.2023,
0.1994, 0.2010])
])
```

Legat de „dropout regularization”, metoda constă în a renunța la o parte din imaginile din setul de date de antrenament în timpul unor pași din antrenament. Pentru implementare am folosit nivelul Dropout din librăria keras, cu parametrul rate având valoarea de 0.2 pentru a specifica la ce procent din date se va renunța.

Similar, în cazul pytorch am folosit nivelul torch.nn.Dropout cu parametrul având valoarea 0.2. Astfel o parte din canalele din tensorul de intrare sunt înlocuite cu valoarea 0 cu probabilitatea 0.2

4.1.6 Crearea modelului

Un pas important în programele ce au la bază rețelele neuronale este proiectarea arhitecturii modelului și apoi implementarea acestuia.

În general, o rețea neuronală este alcătuită din 3 sau mai multe nivele și anume:

- Nivelul de intrare
- Nivelul de procesare, poate fi unul sau mai multe
- Nivelul de ieșire

Nivelul de procesare are mai multe componente și anume:

- Convoluția - extrage părți ale hărții de caracteristici de intrare și aplică filtre pentru a calcula caracteristici noi, producând o hartă de caracteristici de ieșire sau o caracteristică convolută (care poate avea o dimensiune și o adâncime diferite decât harta caracteristică de intrare). Convoluțiile sunt definite de doi parametri:
 - o Dimensiunea bucăților din imagine care sunt extrase (de obicei 3x3 sau 5x5 pixeli).
 - o Adâncimea hărții caracteristicilor de ieșire, care corespunde numărului de filtre care sunt aplicate.
- ReLu - După fiecare operație de convoluție, CNN aplică o transformare Rectified Linear Unit (ReLU) caracteristicii convolutive, pentru a introduce neliniaritatea în model.
- Nivelul de pooling - max pooling funcționează într-un mod similar cu convoluția. Glisăm peste harta caracteristicilor și extragem bucăți de o dimensiune specificată. Pentru fiecare bucată, valoarea maximă este calculată pe o nouă hartă a caracteristicilor și toate celelalte valori sunt eliminate. Operațiile de max-pooling au doi parametri:
 - o Dimensiunea filtrului max-pooling (de obicei 2x2)
 - o Distanța, în pixeli, care separă fiecare bucată extrasă. Spre deosebire de convoluție, în care filtrele se deplasează peste harta caracteristică pixel cu pixel, în max-pooling, pasul determină locațiile în care este extrasă fiecare bucată. Pentru un filtru 2x2, un pas de 2 specifică faptul că operația de max-pooling va extrage toate bucățile 2x2 care nu se suprapun din harta caracteristicilor.

- Dropout – folosit pentru reducerea șanselor de overfitting. o parte din canalele din tensorul de intrare sunt înlocuite cu valoarea 0 cu probabilitatea 0.2
- Flatten - aplatizează dimensiunile tensorilor.
- Dense - în orice rețea neuronală, un strat Dense este un strat care este profund conectat cu stratul său precedent, ceea ce înseamnă că neuronii stratului sunt conectați la fiecare neuron al stratului său precedent. Acest strat este cel mai utilizat strat în rețelele de rețele neuronale artificiale.

O prima arhitectură încercată este cea din figura 13.

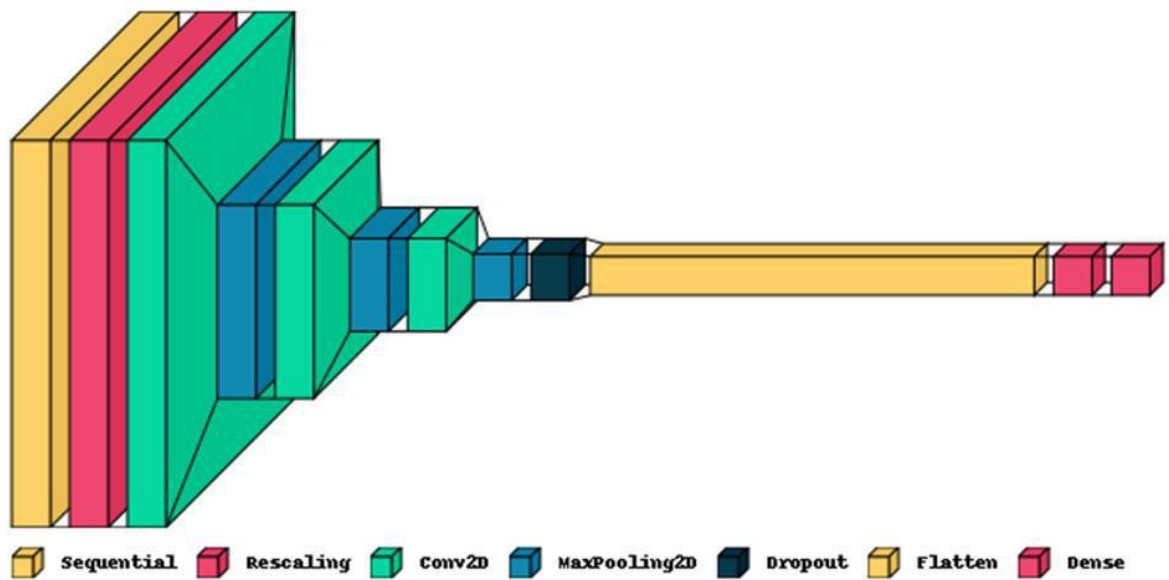


Figura 13 Arhitectura utilizată pentru CNN

Rețeaua are 3 nivele de convoluție, 3 nivele max-pooling, 1 nivel linear și o funcție sigmoidă.

Diagrama arhitecturii modelului din figura 13 este generată utilizând librăria visualekeras din python.

Nivelul de intrare pornește cu imagini de dimensiune $[50,50,3]$ – unde 50 este lățimea și lungimea fiecărei imagini, iar 3 este numărul de canale care inițial corespunde rgb.

Am adăugat un nivel de redimensionare a imaginilor, iar după acest nivel dimensiunea se păstrează ca fiind $[50,50,3]$.

Urmează primul nivel de convoluție cu funcția de activare ReLu, la finalul căruia numărul de canale va deveni 16.

Următorul este un nivel de max-pooling în urma căruia se schimbă și dimensiunea de intrare.

Apoi vom avea un nou nivel de convoluție, după care numărul de canale va deveni 32.

Următorul este un nivel de max-pooling în urma căruia se schimbă și dimensiunea de intrare.

Vom mai adăuga încă un nivel de convoluție, care va transforma numărul de canale în 64, apoi încă un nivel de max-pooling.

Aplicăm un dropout pentru a evita overfittingul, urmat de un nivel de aplatizare.

La final adăugăm două nivele conectate adânc, ultimul din ele având funcția de activare sigmoidă, care este de obicei utilizată în cazul clasificării binare. Dimensiunea rezultatului va fi egală cu 1, iar rezultatul va fi probabilitatea ca imaginea să facă parte din clasa 1.

Pentru implementarea propriu-zisă a modelului în keras, am utilizat clasa Sequential care grupează mai multe nivele. Apoi pentru adăugarea nivelelor am folosit:

- Rescaling cu parametru dimensiunea imaginii și numărul inițial de canale ca fiind 3 și pentru a înlocui fiecare element din matricea imaginii la un element cu valoarea în intervalul [0, 1]
- Conv2D - Acest strat creează un nucleu de convoluție care este convoluat cu intrarea stratului pentru a produce un tensor de ieșire. Dimensiunea nucleului este de 3x3, pasul cu care acesta se deplasează este de 1. Numărul de canale din tensorul de ieșire va fi pe rând 16, 32, 64.

Codul pentru definirea modelului în keras este:

```
self.model = Sequential([
    self.data_augmentation,
    layers.Rescaling(1. / 255, input_shape=(self.IMAGE_SIZE,
self.IMAGE_SIZE, 3)),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.2),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])
```

Codul pentru definirea modelului în pytorch este:

```
class ConvNeuralNet(nn.Module):
    def __init__(self, num_classes):
```

```

super(ConvNeuralNet, self).__init__()
self.conv1 = nn.Conv2d(3, 16, kernel_size=3)
self.conv2 = nn.Conv2d(16, 32, kernel_size=3)
self.conv3 = nn.Conv2d(32, 64, kernel_size=3)
self.pool = nn.MaxPool2d(2, 2)
self.relu = nn.ReLU()
self.fc1 = nn.Linear(1024, 128)
self.fc2 = nn.Linear(128, 1)
self.sigmoid = nn.Sigmoid()

def forward(self, x):
    x = self.conv1(x)
    x = self.relu(x)
    x = self.pool(x)
    x = self.conv2(x)
    x = self.relu(x)
    x = self.pool(x)
    x = self.conv3(x)
    x = self.relu(x)
    x = self.pool(x)
    x = x.view(-1, self.flat_features(x))
    x = self.fc1(x)
    x = self.relu(x)
    x = self.fc2(x)
    x = self.relu(x)
    x = self.sigmoid(x)
    return x

def flat_features(self, x):
    size = x.size()[1:]
    num_features = 1
    for s in size:
        num_features *= s
    return num_features

```

O întrebare interesantă este cum se face schimbarea interpretării dimensiunii canalelor imaginii după ce cel inițial este transformat de primul nivel de convoluție.

Presupunem că avem un tensor de dimensiune 50x50x1 care este trecut prin CNN și ajunge la primul nivel de convoluție. Când se întâmplă acest lucru, forma tensorului și datele vor fi schimbate de operația de convoluție. Convoluția schimbă înălțimea și lungimea imaginii și, de asemenea, numărul de canale. Numărul de canale de ieșire este schimbat în funcție de numărul de filtre folosite în nivelul de convoluție.

Presupunând că avem trei filtre de convoluție, vom avea trei canale de ieșire din nivelul de convoluție. Fiecare dintre cele trei filtre va transforma canalul inițial producând trei canale de ieșire. Canalele de ieșire vor fi în continuare compuse din pixeli, dar aceștia vor fi modificați de operația de convoluție. În funcție de dimensiunea filtrului, lungimea și înălțimea ieșirii se vor schimba și ele.

Canalele de ieșire nu vor mai reprezenta canalele de culoare, ci vor fi numite hațuri de caracteristici. Aceste hațuri de caracteristici sunt rezultatul convoluției care are loc între canalele de culoare și filtrele de convoluție. Cuvântul caracteristică este folosit pentru ca reprezintă o anumită caracteristică a imaginii inițiale cum ar fi spre exemplu muchii, iar aceste mapări rezulta cu cât învățarea în rețea înaintază în timpul antrenamentului și devine din ce în ce mai complex.

4.1.7 Antrenarea

Antrenarea rețelei neuronale se face prin determinarea diferenței dintre rezultatul calculat de către rețea și anume predicția și rezultatul real, pe un set de date de antrenament. Aceasta diferență poartă denumirea de eroare. Inițial fiecare nod din rețea are o pondere. Aceste ponderi se ajustează la fiecare iterație pe baza unei reguli de învățare pe baza erorii de la pasul anterior. La fiecare iterație, după ajustarea ponderilor, eroarea ar trebui să scadă, iar în final rezultatul calculat să fie tot mai aproape de rezultatul real. După un număr suficient de mare de iterații, se poate considera că faza de antrenare s-a încheiat, iar modelul poate produce predicții cu un grad decent de acuratețe.

Dezvoltarea unui model necesită date istorice din domeniul care este utilizat ca date de antrenament. Aceste date sunt compuse din observații sau exemple din domeniul cu elemente de intrare care descriu condițiile și un element de ieșire care surprinde ceea ce înseamnă observația.

Un model de rețea neuronală folosește exemplele pentru a învăța cum să mapeze seturi specifice de variabile de intrare la variabila de ieșire. Trebuie să facă acest lucru în așa fel încât această mapare să funcționeze bine pentru setul de date de antrenament, dar să funcționeze bine și pe exemple noi care nu sunt văzute de model în timpul antrenamentului. Această capacitate de a lucra bine pe exemple specifice și exemple noi se numește generalizare. Putem descrie relația dintre variabilele de intrare și variabilele de ieșire ca o funcție matematică complexă. Pentru o anumită problemă, trebuie să credem că există o funcție de mapare adevărată pentru a mapa cel mai bine variabilele de intrare la variabilele de ieșire și că un model de rețea neuronală poate face o estimare rezonabilă la aproximarea funcției de mapare.

Procesul de antrenare a rețelelor neuronale este cea mai dificilă parte a algoritmului și este de departe cea mai consumatoare de timp, atât în ceea ce privește efortul necesar pentru implementarea modelului, cât și complexitatea de calcul necesară executării procesului.

Trebuie aleasă o funcție de eroare, numită adesea funcție obiectiv, funcție de cost sau funcție de pierdere. Funcțiile de pierdere alese de obicei sunt cross-entropy sau binary cross-entropy pentru problemele de clasificare și eroarea medie pătratică pentru problemele de regresie.

Pentru calcularea pierderii am folosit clasa BinaryCrossentropy, care este de obicei utilizată când există două clase și ne așteptăm ca etichetele să fie furnizate ca numere.

Procesul de căutare sau optimizare necesită un punct de plecare de la care să înceapă actualizările modelului. Punctul de plecare este definit de ponderile inițiale ale modelului. De obicei, sunt alese ca ponderi inițiale ale modelului valori aleatoare mici.

Pentru adaptarea ponderilor am folosit algoritmul Adam, care combină avantajele altor două extensii ale coborârii gradientului stocastic

- Algoritmul Adaptive Gradient (AdaGrad) care menține o rată de învățare pe parametru care îmbunătățește performanța în problemele precum probleme de limbaj natural și de computer vision
- Root Mean Square Propagation (RMSProp) care menține o rată de învățare pe parametru, dar care este adaptată pe baza a cât de repede se schimbă ponderile.

La antrenarea modelului, trebuie folosite o serie de exemple din setul de date de antrenament pentru a calcula eroarea modelului. Pot fi utilizate toate exemplele din setul de date de antrenament, ceea ce poate fi adecvat pentru seturi de date mai mici. Alternativ, poate fi utilizat un singur exemplu care poate fi adecvat pentru problemele în care exemplele sunt transmise în flux sau în care datele se modifică des. O abordare hibridă poate fi utilizată în cazul în care numărul de exemple din setul de date de antrenament poate fi ales și utilizat pentru a estima gradientul de eroare. Alegerea numărului de exemple este denumită dimensiunea lotului.

Am folosit o dimensiune a lotului de 64.

Odată ce un gradient de eroare a fost estimat, eroarea poate fi calculată și utilizată pentru a actualiza fiecare pondere. Adâncimea modelului (numărul de nivele) și faptul că parametrii modelului sunt actualizați separat înseamnă că este greu de calculat exact cât de mult trebuie schimbat fiecare pondere. În schimb, o mică parte din actualizarea ponderilor este efectuată la fiecare iterație. Un hiperparametru numit „rata de învățare” controlează cât de mult să actualizeze greutatea modelului și, la rândul său, controlează cât de repede învață un model pe setul de date de antrenament.

Am folosit rata de învățare egală cu 0.001.

Procesul de antrenament trebuie repetat de multe ori până când este descoperit un set bun sau suficient de bun de ponderi ai modelului. Numărul total de iterații ale procesului este dat de numărul de treceri complete prin setul de date de antrenament, după care procesul de antrenament este încheiat. Acesta este denumit numărul de „epoci” de antrenament.

Am folosit un număr de 5 epoci pentru antrenament.

Pentru antrenarea modelului folosind keras și tensorflow am utilizat funcția `fit` din clasa modelului având ca parametrii setul de antrenament, dimensiunea lotului (batch size), numărul de epoci, dar și setul de date pe care se face validarea.

În keras, implementarea arată în felul următor:

```
self.model.compile(optimizer=tf.keras.optimizers.Adam(),
loss=tf.keras.losses.BinaryCrossentropy(from_logits=False),
metrics=['accuracy',
tf.keras.metrics.Precision(name='precision'),
tf.keras.metrics.Recall(name='recall'),
tf.keras.metrics.SpecifictyAtSensitivity(0.5,
name='specificity_at_sensitivity')])
```

În pytorch, am implementat faza de antrenare după următorul algoritm:

- Pentru fiecare epocă
 - o Pentru fiecare pereche imagine-etichetă din setul de date de antrenament
 - Trecem datele de intrare prin rețea și calculăm rezultatul de ieșire
 - Calculăm funcția de pierdere în funcție de rezultatul calculat și eticheta reală
 - Calculăm propagarea inversa a erorii. Având în vedere rețeaua neuronală și funcția de eroare, metoda calculează gradientul funcției de eroare în raport cu ponderile rețelei neuronale.
 - Pasul de optimizare folosind Adam
 - Verificam dacă predicția clasei a fost sau nu corectă
 - o Calculăm acuratețea, precizia, reapelul, specificitatea și funcția de pierdere pentru epocă

```
- for e in range(self.EPOCHS):
    self.model.train()
    # define the loss value after the epoch
    loss = 0.0
    number_of_sub_epoch = 0
    start = time.time()
    # loop for every training batch (one epoch)
    for images, labels in self.train_dl:
        # in pytorch you have assign the zero for gradien in any sub
```

```

epoch
    self.optimizer.zero_grad()
    # create the output from the network
    out = self.model(images)
    _, predicted = out.max(1)
    # count the loss function
    loss = self.criterion(out, labels.unsqueeze(1).float())
    # count the backpropagation
    loss.backward()
    # learning
    self.optimizer.step()
    # add new value to the main loss
    losss += loss.item()
    number_of_sub_epoch += 1
    self.TP, self.FP, self.TN, self.FN =
self.update_confusionmatrix(out, labels, self.TP, self.FP, self.TN,
self.FN)
    end = time.time()
    print("Epoch {}: Loss: {} Time: {} ".format(e, losss /
number_of_sub_epoch, (end - start)))
    print("TP: {} FP: {} TN: {} FN: {}".format(self.TP, self.FP,
self.TN, self.FN))
    accuracy, precision, recall, specificity =
self.compute_measurements(self.TP, self.FP, self.TN, self.FN)

    print("Accuracy: {} Precision: {} Recall: {} Specificity: {}
".format(accuracy,
precision,
recall,
specificity))
    self.TP = 0
    self.FP = 0
    self.TN = 0
    self.FN = 0
    # torch.save(self.model.state_dict(),
    #
    "E:/automatica/master_an2/Disertatie/Breast_xray/breast_pytorch_cnn_i
ntermediar_epocal.pth")
    self.validate(e)

```

4.1.8 Validarea

Validarea este dificil de făcut corect deoarece presupune predicția pe seturi de date necunoscute modelului astfel încât să putem afla ce funcționează cel mai bine pentru diferite scenarii pentru a determina ce să modificăm în modul în care modelul caută aceste informații.

Setul de date de validare este un set de date pe care dorim ca modelul nostru să aibă preziceri bune, în timpul antrenamentului, adaptăm hiperparametri astfel încât modelul să aibă performanța crescută pe setul de date de validare.

Cu un set de validare luăm o fracțiune din mostrele din setul total și folosim mostrele din acest set de la antrenament.

În fiecare epocă, modelul va fi antrenat pe mostre din setul de antrenament, dar nu va fi antrenat pe mostre din setul de validare. În schimb, modelul va fi validat numai pentru fiecare probă din setul de validare.

Scopul acestui lucru este de a evalua cât de bine se poate generaliza modelul, adică, cât de bine este modelul capabil să prezică datele pe care nu le vede în timpul antrenamentului.

Prezența unui set de validare oferă o perspectivă excelentă dacă modelul este supra adaptat sau nu (overfitting). Acest lucru poate fi interpretat comparând acuratețea și pierderea din eșantioanele de antrenament cu acuratețea și pierderea din mostrele de validare. De exemplu, dacă acuratețea din timpul antrenamentului este mare, dar acuratețea din timpul validării rămâne cu mult în urmă, acesta este un indiciu că modelul este supra adaptat.

În Keras, validarea se face odată cu antrenamentul prin funcția `fit` a modelului căreia îi dam ca parametru `validation_data` exact setul de date de validare.

În cazul PyTorch, validarea se face conform algoritmului următor:

- Se trece modelul în starea de validare utilizând funcția `eval`, care este un tip de comutator pentru anumite părți ale modelului care acționează diferit în timpul antrenamentului și în timpul validării.
- Pentru fiecare pereche imagine-etichetă din setul de date de validare
 - o Trece datele de intrare prin rețea și calculăm rezultatul de ieșire
 - o Verificăm dacă predicția clasei a fost sau nu corectă
- Calculăm acuratețea, precizia, reapelul, specificitatea și pierderea.

```
- def validate(self, e):  
    # PyTorch - Comparing the Results  
    total = 0  
    val_loss = 0.0  
    val_losss = 0.0  
    number_of_sub_epoch = 0  
    self.model.eval()
```

```

        # self.optimizer.zero_grad()
        with torch.no_grad():
            for images, labels in self.test_dl:
                outputs = self.model(images)
                _, predicted = outputs.max(1)
                val_loss = self.criterion(outputs,
labels.unsqueeze(1).float())
                number_of_sub_epoch += 1
                total += labels.size(0)
                val_lossss += val_loss.item()
                # print(outputs)
                self.val_TP, self.val_FP, self.val_TN, self.val_FN =
self.update_confusionmatrix(outputs, labels,

self.val_TP,

self.val_FP,

self.val_TN,

self.val_FN)
                accuracy, precision, recall, specificity =
self.compute_measurements(self.val_TP, self.val_FP, self.val_TN,

self.val_FN)
                print('On {} test images: Val_Loss {}% with
PyTorch'.format(total, val_lossss / number_of_sub_epoch))
                print("Validation TP: {} FP: {} TN: {} FN:
{}".format(self.val_TP, self.val_FP, self.val_TN, self.val_FN))
                print("Val_Accuracy: {} Val_Precision: {} Val_Recall: {}
Val_Specificity: {} ".format(accuracy,

precision,

recall,

specificity))
                self.val_TP = 0
                self.val_FP = 0
                self.val_TN = 0
                self.val_FN = 0

```

4.1.9 Salvarea modelului și încărcarea modelului

Progresul modelului poate fi salvat în timpul și după antrenament. Asta înseamnă că un model poate relua de unde a rămas și poate evita timpii lungi de antrenament. Salvarea înseamnă, de asemenea, că putem partaja modelul. Atunci când publică modele și tehnici de cercetare, majoritatea practicienilor de învățare automată împărtășesc:

- cod pentru a crea modelul
- ponderile antrenate, sau parametrii, pentru model

În keras, am salvat modelul folosind funcția `save` ce conține ca parametru calea unde va fi salvat modelul. De asemenea, pentru încărcare am folosit funcția `load` care conține ca parametru calea de unde va fi încărcat modelul.

În pytorch am folosit funcția `torch.save` ce are ca parametrii modelul și calea unde va fi salvat modelul. De asemenea, pentru încărcare am folosit funcția `torch.load` care conține ca parametru calea de unde va fi încărcat modelul și returnează modelul. O altă opțiune de salvare este aceea de a salva doar parametrii învățați ai modelului antrenat. În PyTorch, parametrii învățabili ai unui model `torch.nn.Module` sunt conținuți în parametrii modelului. Un `state_dict` este pur și simplu un obiect dicționar Python care mapează fiecare strat la tensorul său de parametri. Numai nivelele cu parametri învățați (straturi de convoluție, straturi liniare etc.) și buffer-uri înregistrate au intrări în `state_dict` al modelului. Obiectele optimizatorului (`torch.optim`) au și ei un `state_dict`, care conține informații despre starea optimizatorului, precum și hiperparametrii utilizați. Deoarece obiectele `state_dict` sunt dicționare Python, ele pot fi ușor salvate, actualizate, modificate și restaurate, adăugând multă modularitate modelelor și optimizatorilor PyTorch. Salvarea `state_dict` a modelului cu funcția `torch.save()` va oferi cea mai mare flexibilitate pentru restaurarea modelului mai târziu, motiv pentru care este metoda recomandată pentru salvarea modelelor. Funcția `load_state_dict()` preia un obiect dicționar, nu o cale către un obiect salvat ceea ce înseamnă că trebuie să deserializăm obiectul `state_dict` salvat înainte de a-l da ca parametru funcției `load_state_dict()`.

4.1.10 Interfața cu utilizatorul

Pentru implementarea interfeței cu utilizatorul am folosit librăria Tkinter. Tkinter vine de la Tk interface și este o legătură între Python și setul de instrumente Tk GUI. Tkinter nu este o singură librărie, ci mai degrabă constă din câteva module distincte, fiecare cu funcționalitate separată și propria documentație oficială.

Pentru crearea ferestrei folosim constructorul `Tk()` care întoarce o fereastră nouă.

Pentru a defini dimensiunile ferestrei am utilizat funcția `geometry` ce are un parametru de forma `widthxheight+x+y` pentru a defini lungimea și lățimea ferestrei, dar și un offset pentru poziție.

Titlul ferestrei se dă utilizând funcția `title()`, iar culoarea de fundal utilizând funcția `background()`.

Odată ce am inițializat fereastra, putem începe să adăugăm elemente. Un prim element de care avem nevoie este un buton ce va fi folosit pentru încărcarea imaginii. Pentru aceasta există clasa `Button`, a-l cărei constructor are ca parametrii

- fereastra în care va fi poziționat și care a fost creată anterior
- un text descriptiv
- o comandă sub forma unei funcții care va fi apelată la apăsare
- poziția în fereastra sub forma coordonatelor x și y

Folosind funcția `configure` putem adăuga butonului culori de fundal, și configura proprietățile scrisului precum dimensiunea sau fontul.

Comanda configurată pentru butonul de încărcare este o funcție implementată manual care citește calea adăugată de utilizator și apoi încarcă imaginea într-un thumbnail. Apoi, această funcție va face vizibil butonul ce va face posibilă operația de clasificare a imaginii. Pentru acest buton am adăugat funcția de clasificare. Această funcție apelează funcția de încărcare a modelului în funcție de algoritmul selectat. Apoi procesează imaginea încărcată, care apoi este trecută prin rețea și la final se face o predicție. Această predicție este apoi afișată utilizatorului printr-un obiect de tip `label`. Un alt buton configurat este cel de adăugare a unui nivel nou la model în cazul funcționalității de antrenare a unui nou model și cel de compilare a modelului, care e folosit pentru antrenarea propriu-zisă.

Următorul element este un grup de butoane implementate prin clasa `Radiobutton` și care au ca valori posibile unul din algoritmi utilizați. De asemenea, am folosit `Radiobutton` pentru a selecta tipul de nivel ce va fi adăugat la model în cazul funcționalității de antrenare a unui nou model.

Alt element folosit este componenta de tip `Entry` care ține locul de `textbox` și e folosită pentru introducerea de date precum calea către date, număr de epoci sau parametrii nivelelor adăugate la model în cazul funcționalității de antrenare a unui nou model.

La final, fereastra este procesată prin intermediul funcției `mainloop` pentru a face posibilă interacțiunea cu utilizatorul prin monitorizarea evenimentelor introduse de acesta în sistem și procesarea răspunsului la aceste evenimente.

4.1.11 Clase și funcții

Diagrama de clase este prezentată în figura 15.

Deoarece toți algoritmi de Deep Learning implementați și comparați au o structură similară, am optat pentru utilizarea unei interfețe. Aceasta interfață este implementată apoi de fiecare dintre celelalte patru clase ce implementează algoritmi comparați. Pentru vizualizare și coordonare am implementat și clasa Gui care reprezintă interacțiunea cu utilizatorul.

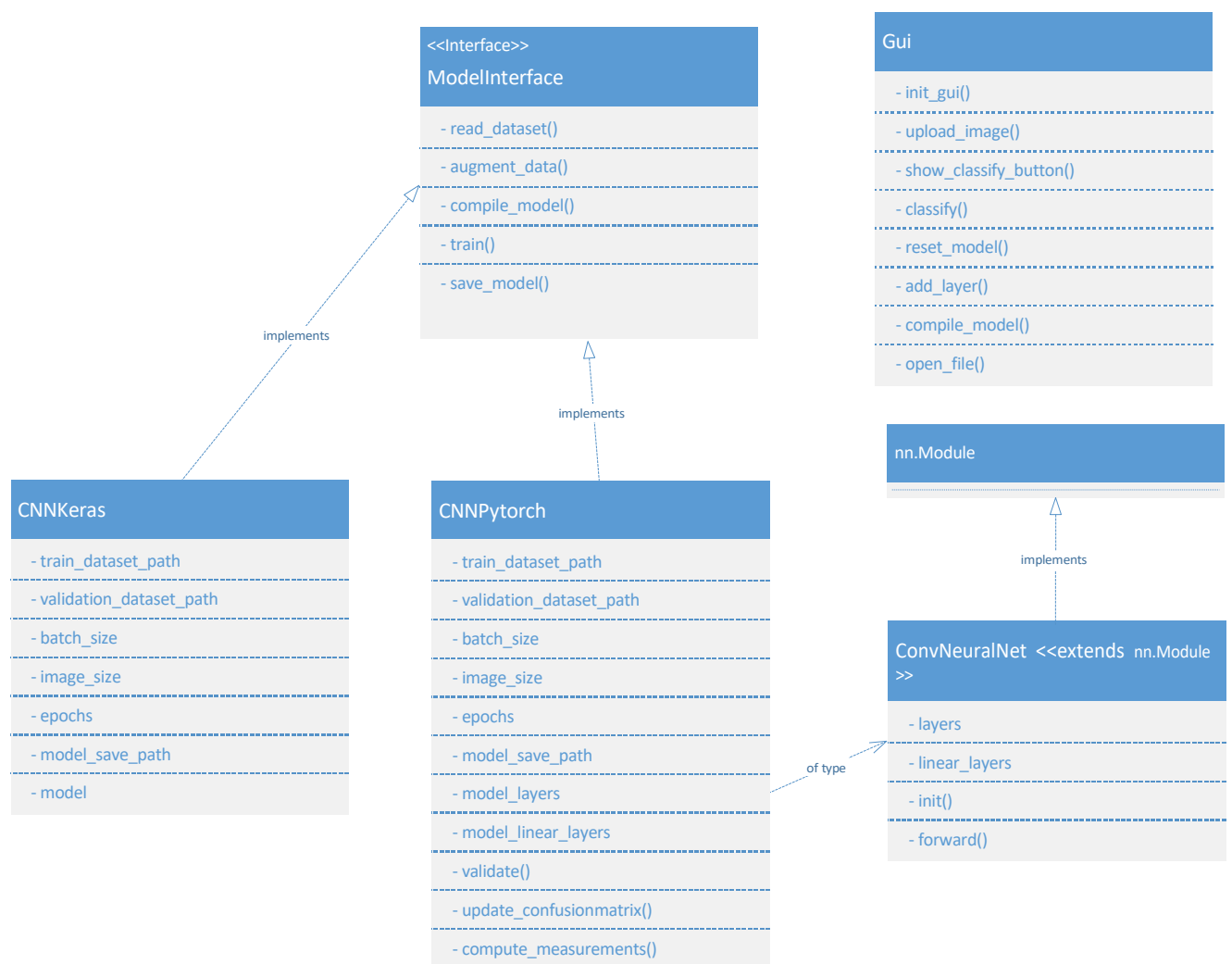


Figura 14 Diagrama de clase

Prototipul funcției	<code>void read_dataset()</code>
---------------------	----------------------------------

Parametrii	N/A
Valoare returnată	N/A
Descriere	Citește seturile de date de antrenament și validare de pe dispozitiv.
Context	main

Prototipul funcției	void augment_data ()
Parametrii	N/A
Valoare returnată	N/A
Descriere	Aplică operații de rotație, scalare și translație asupra setului de date cu un grad de rotație și scalare aleatoriu și translație aleatorie pe axa verticală sau orizontală.
Context	main

Prototipul funcției	void compile_model()
Parametrii	N/A
Valoare returnată	N/A
Descriere	Creează o instanță a modelului cu arhitectura data, creează funcția de pierdere și optimizatorul și apoi compilează modelul.
Context	main

Prototipul funcției	void train()
Parametrii	N/A
Valoare returnată	N/A

Descriere	<p>Funcția folosită pentru antrenarea modelului. În cazul keras apelează funcția fit aparținând clasei modelului și îi dă ca parametru numărul de epoci, dimensiunea lotului, dar și seturile de date de antrenament și validare. în cazul keras, aici se execută și validarea în contextul metodei fit.</p> <p>În cazul pytorch, se executa doar pașii de antrenament, iar aceștia sunt implementați manual conform capitolului 4.1.6.</p>
Context	main

Prototipul funcției	void validate()
Parametrii	N/A
Valoare returnată	N/A
Descriere	<p>Aplicabilă doar în contextul pytorch, consta în etapa de validare. Setul de date de validare este iterat de un număr de ori egal cu numărul de epoci și fiecare lot de imagini din setul de date este trecut prin model. Se calculează pierderea, acuratețea, precizia, reapelul și specificitatea, dar ponderile din model nu sunt actualizate.</p>
Context	train

Prototipul funcției	loc_TP, loc_FP, loc_TN, loc_FN update_confusionmatrix(y_actual, y_hat, loc_TP, loc_FP, loc_TN, loc_FN)
Parametrii	<p>y_actual – valoarea calculată de către model pentru lotul din setul de date curent. Reprezintă probabilitatea pentru fiecare imagine din lot sa aparțină clasei cu id 1.</p> <p>y_hat – valoarea reală a clasei de care aparține fiecare imagine din lot.</p> <p>loc_TP – valoare calculată până la momentul curent pentru toate rezultatele în care modelul a prezis clasa 1 și clasa 1 era cea reală.</p>

	<p>loc_FP – valoare calculată până la momentul curent pentru toate rezultatele în care modelul a prezis clasa 1 și clasa 0 era cea reală.</p> <p>loc_TN – valoare calculată până la momentul curent pentru toate rezultatele în care modelul a prezis clasa 0 și clasa 0 era cea reală.</p> <p>loc_FN - valoare calculată până la momentul curent pentru toate rezultatele în care modelul a prezis clasa 0 și clasa 1 era cea reală.</p>
Valoare returnată	<p>loc_TP – valoare actualizată a parametrului de intrare.</p> <p>loc_FP – valoare actualizată a parametrului de intrare.</p> <p>loc_TN – valoare actualizată a parametrului de intrare.</p> <p>loc_FN - valoare actualizată a parametrului de intrare.</p>
Descriere	Aplicabilă doar în contextul pytorch, metoda este folosită pentru calcularea elementelor din matricea de confuzie (TP, FP, TN, FN), care mai departe vor fi folosite la calcularea metricilor de acuratețe, precizie, reapel și specificitate.
Context	train, validate

Prototipul funcției	Accuracy, precision, recall, specificity update_confusionmatrix(loc_TP, loc_FP, loc_TN, loc_FN)
Parametrii	<p>loc_TP – valoare calculată până la momentul curent pentru toate rezultatele în care modelul a prezis clasa 1 și clasa 1 era cea reală.</p> <p>loc_FP – valoare calculată până la momentul curent pentru toate rezultatele în care modelul a prezis clasa 1 și clasa 0 era cea reală.</p> <p>loc_TN – valoare calculată până la momentul curent pentru toate rezultatele în care modelul a prezis clasa 0 și clasa 0 era cea reală.</p> <p>loc_FN - valoare calculată până la momentul curent pentru toate rezultatele în care modelul a prezis clasa 0 și clasa 1 era cea reală.</p>

Valoare returnată	<p>accuracy – procentul de valori prezise corect din total</p> <p>precision – procentul de 1 prezis corect din toate prezicerile de 1 făcute</p> <p>recall– procentul de 1 prezis corect din toate valorile reale de 1</p> <p>specificity - procentul de 0 prezis corect din toate valorile reale de 0</p>
Descriere	Aplicabilă doar în contextul pytorch, metoda este folosită pentru calcularea metricilor folosite în evaluarea performanței și anume acuratețe, precizie, reapel și specificitate.
Context	train, validate

Prototipul funcției	save_model()
Parametrii	N/A
Valoare returnată	N/A
Descriere	Folosită pentru salvarea modelului după ce acesta a fost antrenat, pentru folosirea ulterioară a acestuia în aplicație.
Context	main

Prototipul funcției	load_model()
Parametrii	N/A
Valoare returnată	N/A
Descriere	Folosită pentru încărcarea modelului salvat anterior, pentru folosirea ulterioară a acestuia în aplicație.
Context	main

Prototipul funcției	init_gui()
---------------------	------------

Parametrii	N/A
Valoare returnată	N/A
Descriere	Folosită pentru crearea interfeței cu utilizatorul.
Context	main

Prototipul funcției	add_layer()
Parametrii	N/A
Valoare returnată	N/A
Descriere	Citește parametrii introduși de utilizator și adaugă un nou nivel la model.
Context	La apăsarea butonului add layer din interfață.

Prototipul funcției	upload_image()
Parametrii	N/A
Valoare returnată	N/A
Descriere	Deschide un explorator de directoare, primește calea pozei selectate de utilizator și afișează imaginea în interfață.
Context	La apăsarea butonului upload din interfață.

Prototipul funcției	classify ()
Parametrii	N/A
Valoare returnată	N/A

Descriere	Trece imaginea selectata prin model și afișează în interfața clasa căreia îi aparține imaginea și gradul de încredere în prezicere.
Context	La apăsarea butonului classify din interfață.

4.1.12 Testare

O fază importantă în dezvoltarea unei aplicații este faza de testare. Fiecare tip de testare are avantajele și dezavantajele sale.

Testarea software presupune execuția unei componente software sau a unei componente a sistemului pentru a evalua comportamentul acesteia. În general, sunt evaluate următoarele proprietăți:

- sistemul întrunește cerințele după care a fost proiectat
- sistemul răspunde corect la toate evenimentele de intrare
- sistemul răspunde intru-un interval de timp rezonabil
- poate fi instalat și executat

Tipul de testare utilizat este testare de sistem, iar în acest caz întreg sistemul este testat conform cerințelor.

Teste:

Cerința testată	Aplicația trebuie să permită selectarea unei imagini de pe dispozitiv.
Evenimente de intrare	Utilizatorul apasă butonul upload și încarcă o imagine.
Rezultat așteptat	Imaginea este cu succes încărcată.
Statusul testului	ok

Cerința testată	Aplicația trebuie să permită clasificarea unei imagini în una din clasele predefinite.
-----------------	--

Evenimente de intrare	Utilizatorul apasă butonul upload și încarcă o imagine. Utilizatorul apasă butonul classify.
Rezultat așteptat	Imaginea este cu succes încărcată. După apăsarea butonului classify, deasupra imaginii apare clasa și gradul de încredere.
Statusul testului	ok

Cerința testată	<ul style="list-style-type: none"> - Aplicația trebuie să permită selectarea algoritmului ce va fi folosit în clasificarea imaginii - Aplicația trebuie să permită afișarea rezultatului clasificării. - Imaginea introdusă pentru a fi clasificată poate aparține unei singure clase din domeniul ales.
Evenimente de intrare	Utilizatorul apasă butonul load model. Utilizatorul alege de pe dispozitiv un model pre antrenat.
Rezultat așteptat	Modelul pre antrenat este încărcat cu succes.
Statusul testului	ok

Cerința testată	<ul style="list-style-type: none"> - Aplicația trebuie să permită colectarea parametrilor necesari pentru definirea arhitecturii unei rețele neuronale convoluționale. - Aplicația trebuie să permită antrenarea pe un set de date binar folosind algoritmul și arhitectura selectată. - Aplicația trebuie să permită afișarea metricilor performanței după terminarea antrenării.
-----------------	---

Evenimente de intrare	Utilizatorul selectează librăria(pytorch sau keras). Utilizatorul completează câmpurile epoca, calea către seturile de date, dimensiunea lotului. Utilizatorul adaugă nivelele din rețea. Utilizatorul apasă butonul train.
Rezultat așteptat	Rețeaua este antrenată. Se afișează acuratețea, precizia, reapelul, specificitatea și pierderea pentru etapele de validare și antrenare din ultima epoca.
Statusul testului	ok

4.1.13 Parametrii ajustabili

Pe lângă testarea algoritmului CNN cu arhitectura din figura 13, am făcut o comparație și cu o rețea convoluționile cu mai puține nivele de convoluție. Astfel, putem lua în calcul rețeaua din figura de mai jos:

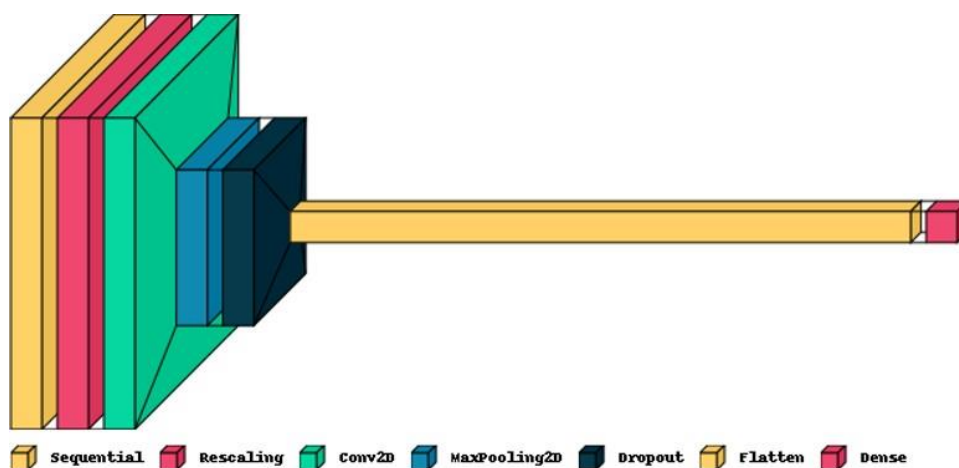


Figura 15 Arhitectura cu 3 nivele

Astfel, rețeaua este formată din:

- 1 nivel de convoluție
- 1 nivel de max-pooling
- 1 nivel linear ce aplică o funcție sigmoidă

4.1.14 Prezentarea rezultatelor

În cazul implementării utilizând keras a algoritmului CNN cu arhitectura din figura 13, am observat rezultatele din graficele următoare:

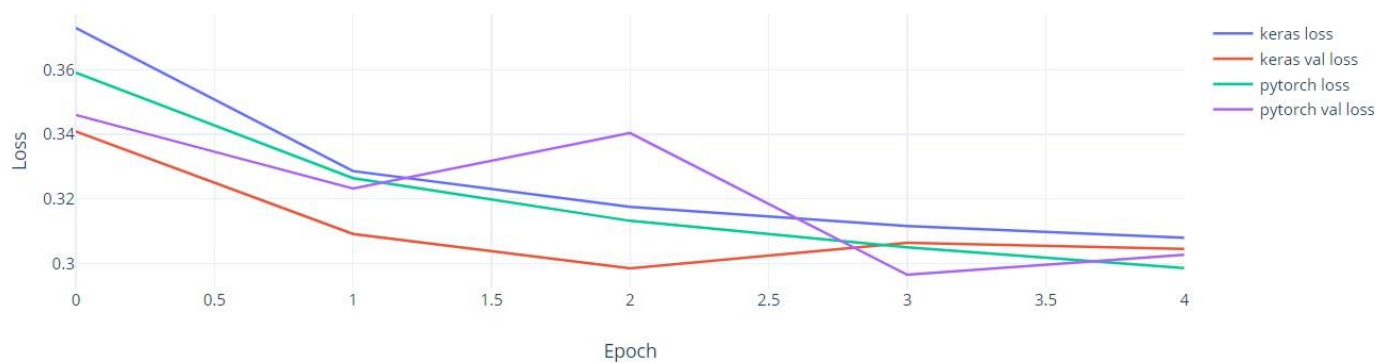


Figura 16 Pierdere pentru CNN cu 8 straturi

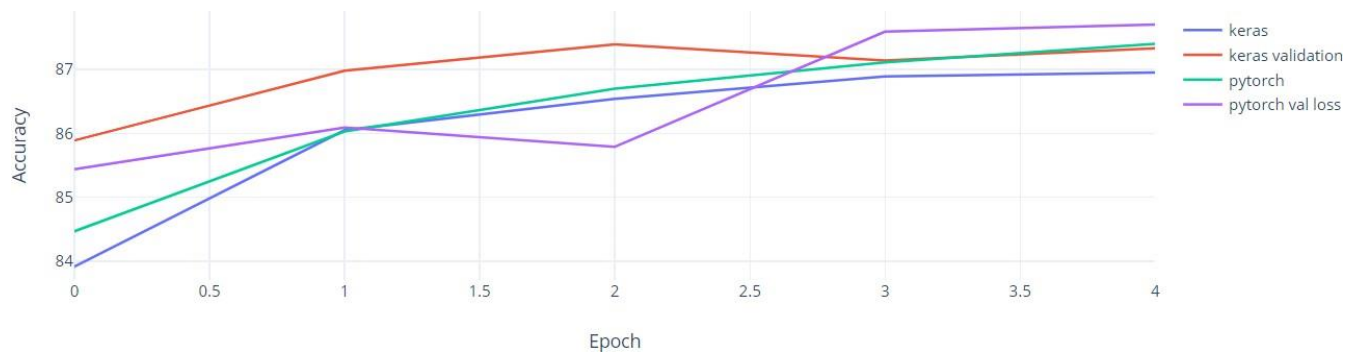


Figura 17 Acuratețe pentru CNN cu 8 straturi

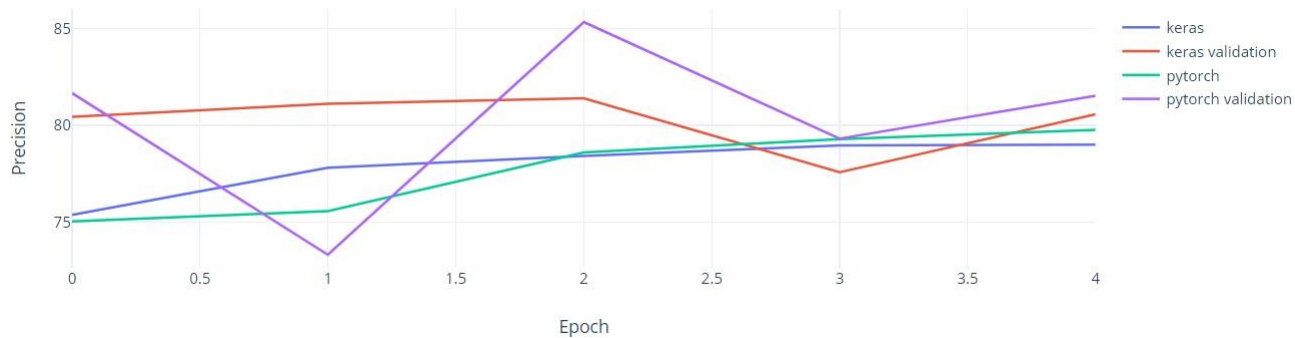


Figura 18 Precizie pentru CNN cu 8 straturi

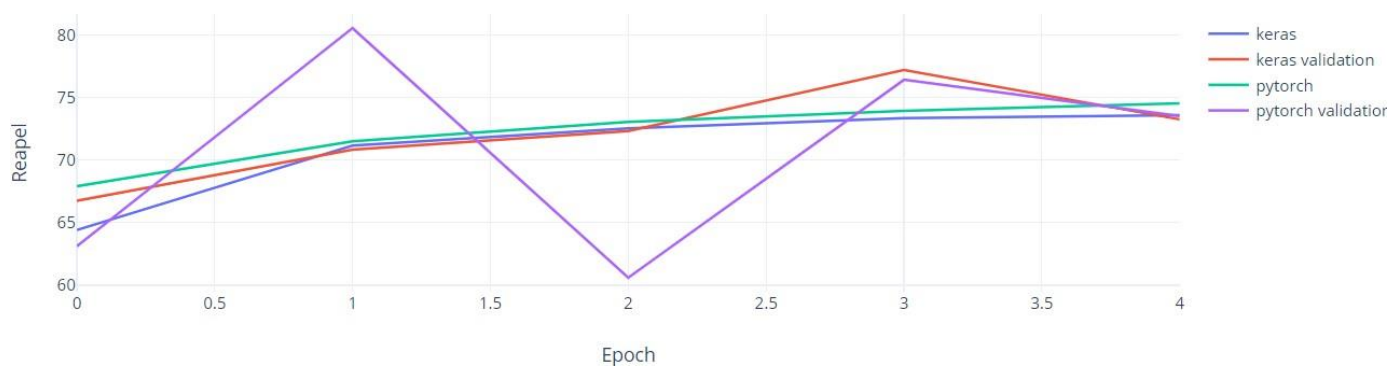


Figura 19 Reapel pentru CNN cu 8 straturi

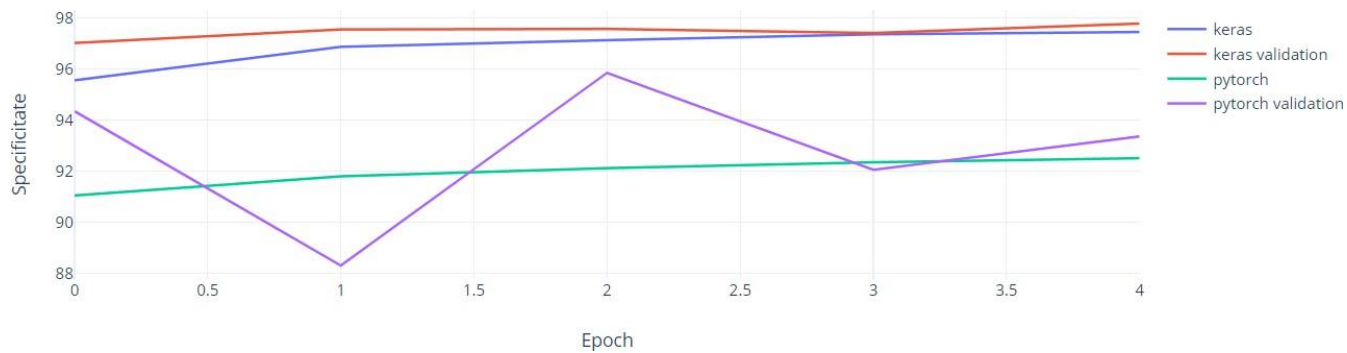


Figura 20 Specificitate pentru CNN cu 8 straturi

O comparație între CNN din keras și CNN din pytorch având aceeași arhitectură este dată în tabelul următor:

Tabel 1 Comparație keras-pytorch cnn antrenament

Epoca	Acuratețe		Precizie		Reapel		Specificitate		Loss	
Librărie	Keras	Pytorch	Keras	Pytorch	Keras	Pytorch	Keras	Pytorch	Keras	Pytorch
0	83.92%	84.47%	75.38%	75.04%	64.42%	67.905	95.55%	91.04%	37.30%	35.91%
1	86.05%	86.03%	77.81%	75.57%	71.17%	71.51%	96.86%	91.79%	32.86%	32.64%
2	86.54%	86.70	78.43%	78.60%	72.54%	73.06%	97.12%	92.11%	31.75%	31.32%
3	86.89%	87.11%	78.97%	79.29%	73.36%	73.94%	97.35%	92.34%	31.16%	30.50%
4	86.95%	87.40%	79.01%	79.77%	73.59%	74.54%	97.44%	92.50%	30.80%	29.86%

Tabel 2 Comparație keras-pytorch cnn validare

Epoca	Acuratețe		Precizie		Reapel		Specificitate		Loss	
Librărie	Keras	Pytorch	Keras	Pytorch	Keras	Pytorch	Keras	Pytorch	Keras	Pytorch
0	85.89%	85.44%	80.45%	81.67%	66.76%	63.12%	97.01%	94.34%	34.09%	34.60%
1	86.98%	86.09%	81.12%	73.32%	70.83%	80.56%	97.54%	88.30%	30.91%	32.32%
2	87.39%	85.79%	81.40%	85.34%	72.33%	60.60%	97.56%	95.84%	29.85%	34.04%
3	87.14%	87.59%	77.59%	79.32%	77.20%	76.42%	97.40%	92.05%	30.64%	29.65%
4	87.33%	87.70%	80.57%	81.53%	73.26%	73.52%	97.77%	93.35%	30.45%	30.27%

Pentru arhitectura din figura 15 avem următoarele rezultate:

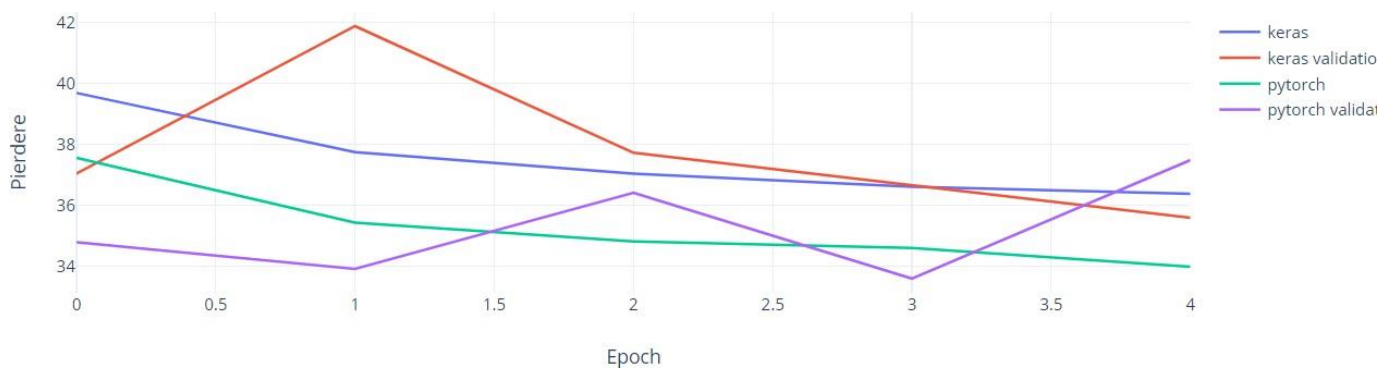


Figura 21 Pierdere pentru CNN cu 3 straturi

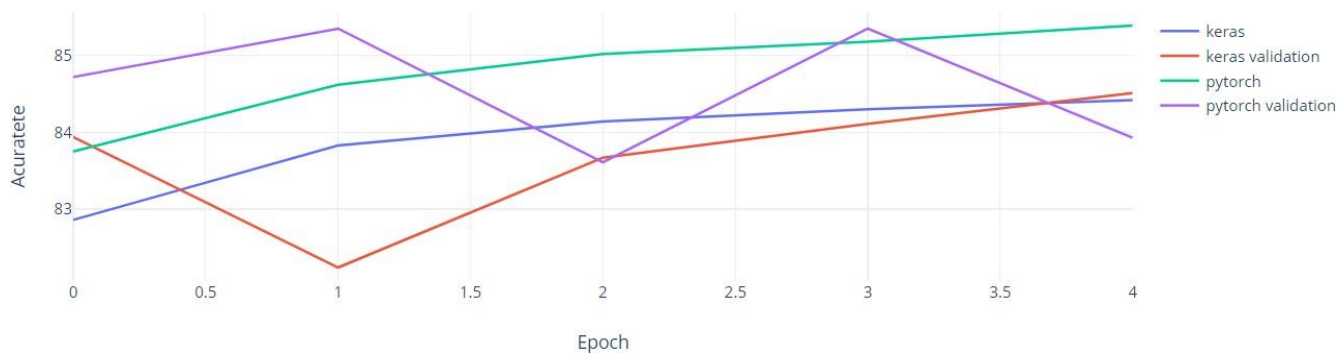


Figura 22 Acuratețe pentru CNN cu 3 straturi

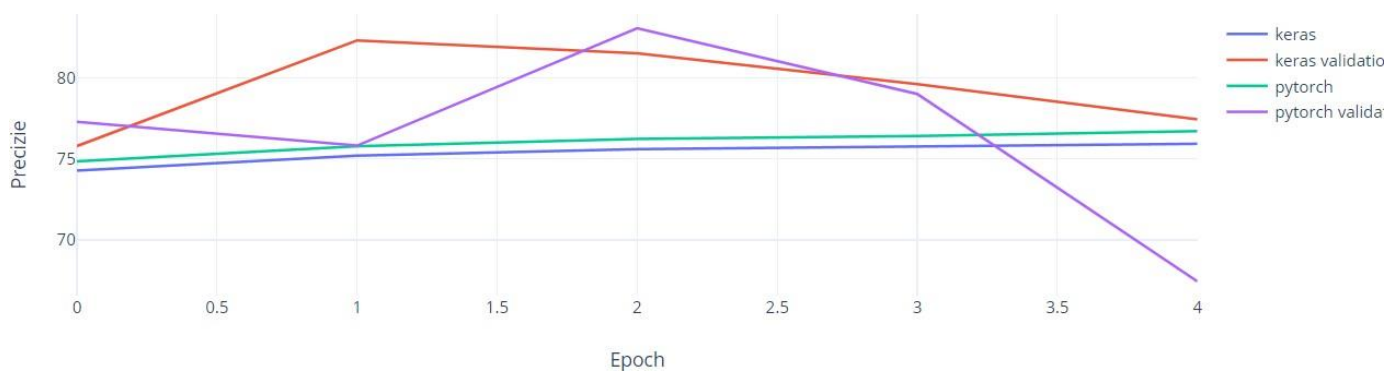


Figura 23 Precizie pentru CNN cu 3 straturi

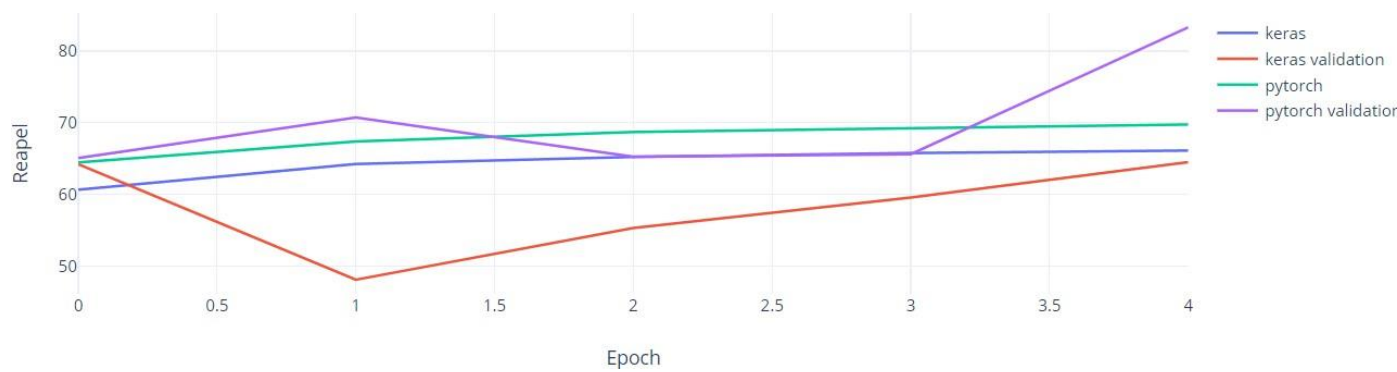


Figura 24 Reapel pentru CNN cu 3 straturi

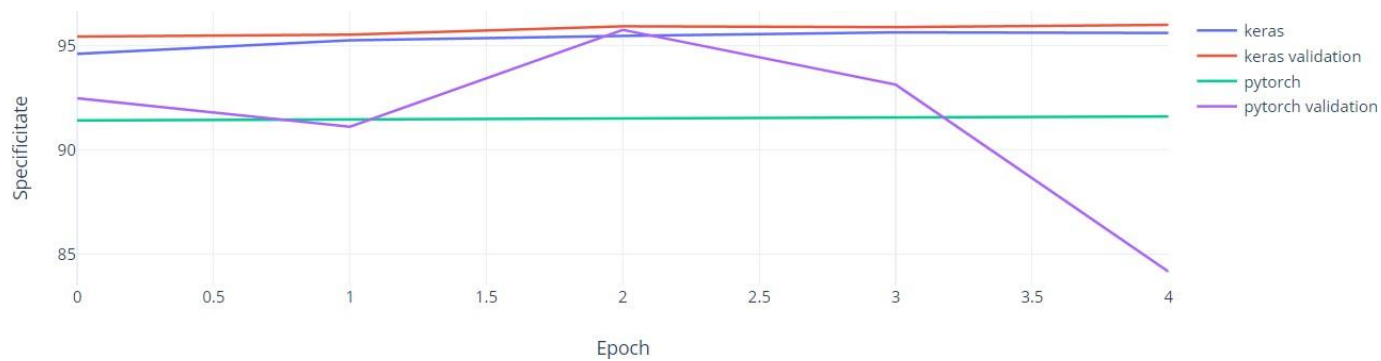


Figura 25 Specificitate pentru CNN cu 3 straturi

Tabel 3 Comparație keras-pytorch 3 nivele antrenament

Epoca	Acuratețe		Precizie		Reapel		Specificitate		Loss	
Librărie	Keras	Pytorch	Keras	Pytorch	Keras	Pytorch	Keras	Pytorch	Keras	Pytorch
0	82.86%	83.75	74.28%	74.85%	60.64%	64.45%	94.60%	91.41%	39.69%	37.56%
1	83.83%	84.62	75.20%	75.78%	64.23%	67.38%	95.25%	91.45%	37.75%	35.44%
2	84.14%	85.02	75.60%	76.23%	65.22%	68.70%	95.46%	91.50%	37.04%	34.82%
3	84.30%	85.18	75.75%	76.41%	65.76%	69.19%	95.63%	91.52%	36.62%	34.61%
4	84.42%	85.39	75.93%	76.71%	66.11%	69.75%	95.60%	91.60%	36.38%	33.99%

Tabel 4 Comparație keras-pytorch 3 nivele validare

Epoca	Acuratețe		Precizie		Reapel		Specificitate		Loss	
Librărie	Keras	Pytorch	Keras	Pytorch	Keras	Pytorch	Keras	Pytorch	Keras	Pytorch
0	83.94%	84.72%	75.79%	77.28%	64.20%	65.06%	95.43%	92.47%	37.05%	34.80%
1	82.24%	85.35%	82.30%	75.82%	48.06%	70.71%	95.52%	91.11%	41.88%	33.92%
2	83.67%	83.61%	81.50%	83.05%	55.28%	65.27%	95.92%	95.75%	37.73%	36.42%
3	84.11%	85.35%	79.61%	79.00%	59.55%	65.61%	95.88%	93.13%	36.65%	33.60%
4	84.51%	83.93%	77.43%	67.45%	64.49%	83.35%	95.99%	84.16%	35.60%	37.49%

Pentru arhitectura din figura 13, dar pe un set de date de dimensiuni reduse, avem următoarele rezultate:

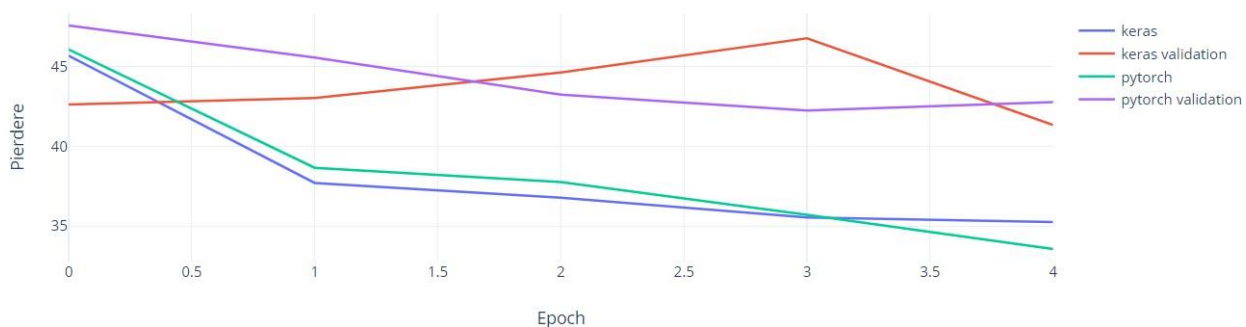


Figura 26 Pierdere set de date mic

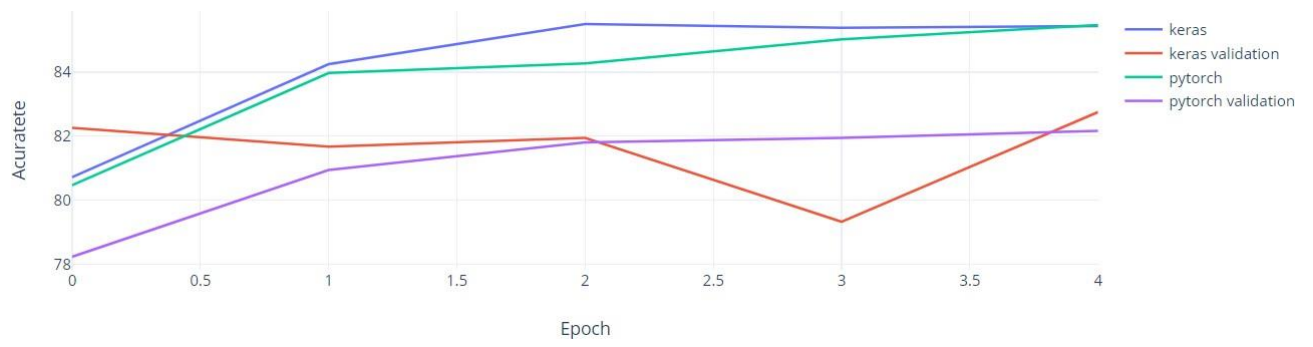


Figura 27 Acuratețe set de date mic

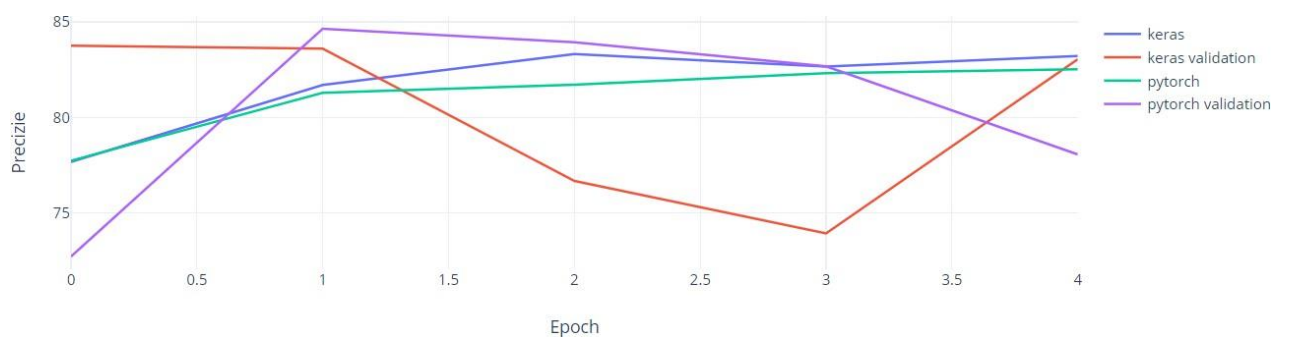


Figura 28 Precizie set de date mic

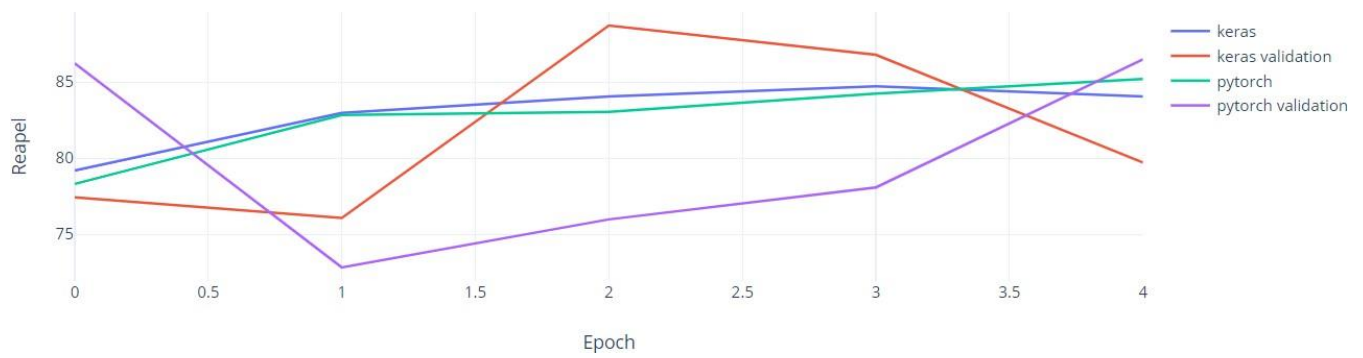


Figura 29 Reapel set de date mic

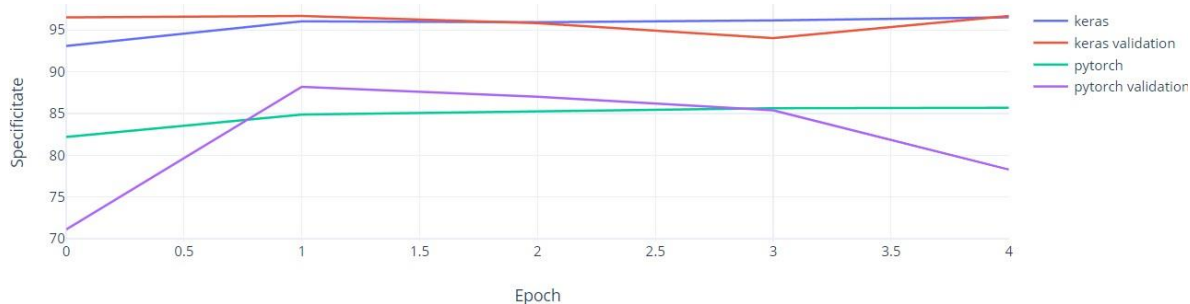


Figura 30 Specificitate set de date mic

Tabel 5 Comparație keras-pytorch 8 nivele antrenament set de date de dimensiune redusa

Epoca	Acuratețe		Precizie		Reapel		Specificitate		Loss	
Librărie	Keras	Pytorch	Keras	Pytorch	Keras	Pytorch	Keras	Pytorch	Keras	Pytorch
0	80.72%	80.47%	77.67%	77.73%	79.19%	78.31%	93.08%	82.19%	45.67%	46.06%
1	84.25%	83.97%	81.70%	81.29%	82.97%	82.83%	96.03%	84.87%	37.70%	38.65%
2	85.50%	84.27%	83.32%	81.71%	84.05%	83.04%	95.92%	85.25%	36.77%	37.76%
3	85.38%	85.02%	82.66%	82.31%	84.73%	84.25%	96.14%	85.63%	35.54%	35.70%
4	85.44%	85.47%	83.21%	82.52%	84.05%	85.20%	96.51%	85.68%	35.25%	33.56%

Tabel 6 Comparație keras-pytorch 8 nivele validare set de date de dimensiune redusa

Epoca	Acuratețe		Precizie		Reapel		Specificitate		Loss	
Librărie	Keras	Pytorch	Keras	Pytorch	Keras	Pytorch	Keras	Pytorch	Keras	Pytorch
0	82.26%	78.23%	83.75%	72.71%	77.42%	86.22%	96.50%	71.11%	42.62%	47.56%
1	81.67%	80.94%	83.60%	84.64%	76.08%	72.82%	96.67%	88.20%	43.02%	45.55%

2	81.94%	81.80%	76.67%	83.93%	88.71%	75.98%	95.81%	87.00%	44.62%	43.23%
3	79.32%	81.94%	73.92%	82.67%	86.79%	78.08%	94.02%	85.38%	46.76%	42.23%
4	82.75%	82.16%	83.05%	78.06%	79.71%	86.50%	96.67%	78.29%	41.33%	42.76%

5 CONCLUZII

Prin lucrarea realizată am comparat același algoritm CNN de clasificare de imagini, utilizând două librării python, și anume keras și pytorch. Metricile calculate pentru a realiza comparația sunt acuratețea, precizia, reapelul, pierderea și specificitatea clasificării. De asemenea, am realizat o comparație și între două variații ale algoritmului CNN, diferența fiind dată de un număr diferit de nivele de convoluție. La final, experimentul a fost realizat pe un set de date de tip binar, din domeniul medical, dimensiunea acestuia fiind de aproximativ 300000 de imagini.

În plus, am pus la dispoziție o aplicație grafică, utilizată pentru clasificarea propriu-zisă a unor imagini. De asemenea, aplicația poate fi folosită pentru antrenarea pe noi seturi de date și definirea de noi rețele neuronale convoluționale.

Lucrarea prezintă în prima parte câteva aspecte teoretice, iar în a doua parte, detalii despre dezvoltarea propriu-zisă a aplicației.

Rezultatele pot fi observate în graficele din figurile 16-30 și tabelele 1-6.

Pe baza experimentelor de mai sus, se poate observa că în majoritatea cazurilor algoritmul implementat în Pytorch are rezultate mai bune.

Pentru rețeaua cu 8 nivele, pytorch a avut rezultate mai mari la acuratețe, precizie, reapel și pierdere, atât în cazul antrenamentului cât și a validării. În schimb, specificitatea este mai mică în ambele cazuri.

Și în cazul rețelei cu 3 nivele, rezultatele la antrenament și validare mai bune au fost tot la algoritmul implementat în pytorch. Specificitatea este din nou mai scăzută în cazul pytorch.

În schimb, pentru experimentul pe setul de date de dimensiuni reduse, pentru etapa de validare, rezultatele mai bune au fost în cazul keras, deși în faza de antrenament pytorch a avut un avantaj. Acest lucru ar putea însemna că algoritmul în pytorch este mai predispus la overfitting.

Un posibilă îmbunătățire ar fi extinderea aplicației pentru a suporta și clasificarea multi-clasă.

6 MANUAL DE UTILIZARE

Interfața cu utilizatorul arată ca în figura de mai jos:

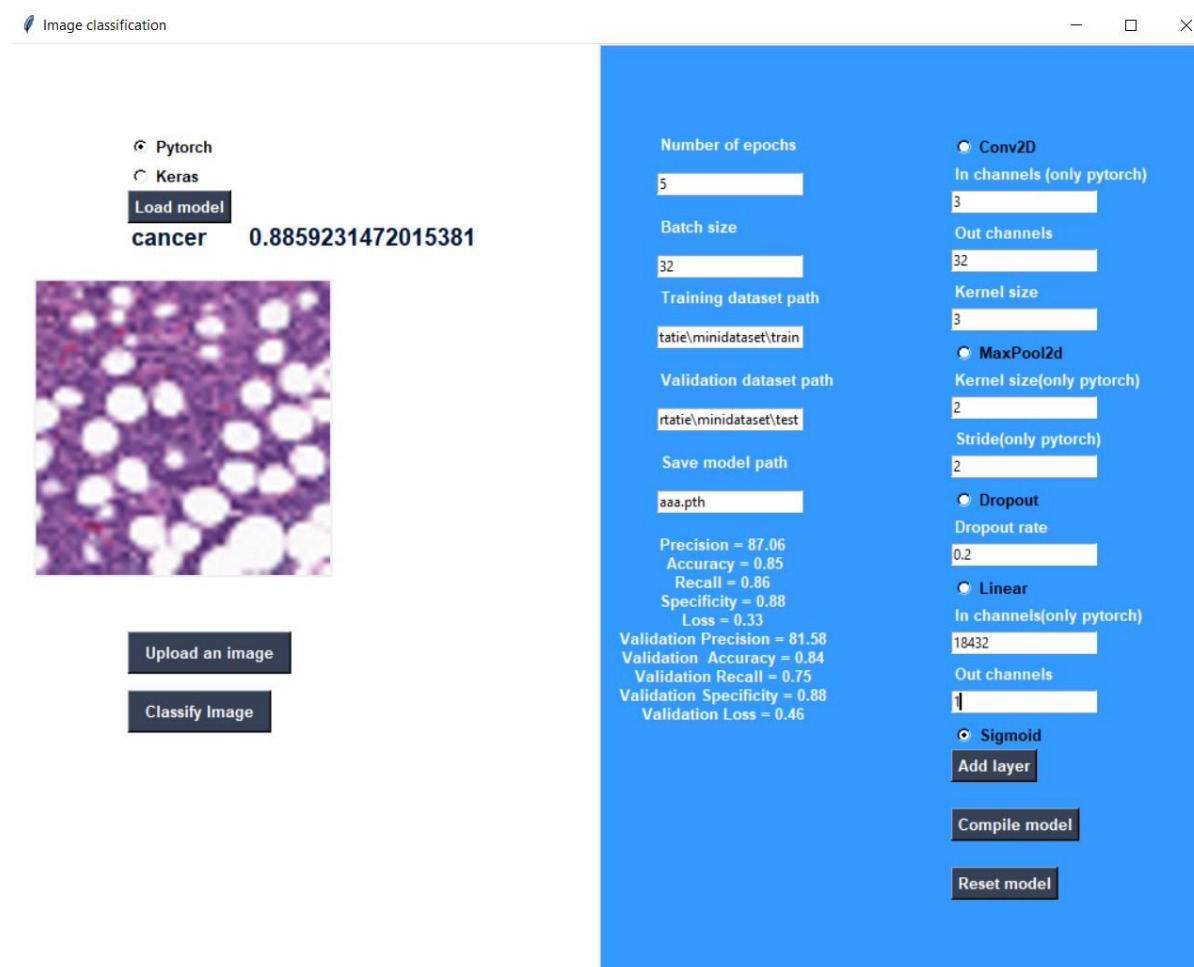


Figura 31 Interfața cu utilizatorul

Pentru a clasifica o imagine trebuie specificată librăria în care va fi implementat (pytorch sau keras) prin intermediul radiobutton-ului. Apoi se apasă pe butonul luda model și se alege calea către modelul deja antrenat. Trebuie avut grijă ca modelul să corespundă cu algoritmul selectat. Apoi se apasă pe butonul Upload an Image și se alege de pe dispozitiv o imagine. Butonul Classify image devine vizibil, iar după apăsarea acestuia vor fi afișate eticheta și gradul de încredere calculat de algoritm deasupra imaginii. Procedul este identic în cazul keras și pytorch.

Pentru crearea și antrenarea unui nou model trebuie specificată librăria în care va fi implementat (pytorch sau keras) prin intermediul radiobutton-ului. Apoi trebuie completate câmpurile din partea albastră a interfeței și anume Number of pochs, Batch size, Training dataset path, Validation dataset path, Save dataset path. În cazul alegerii căii unde va fi salvat modelul, trebuie ținut cont de extensia modelului (pth pentru pytorch sau h5 pentru keras). Apoi se adaugă, pe rând, nivele pentru a crea modelul. Pot fi adăugate oricât de multe nivele, dar, desigur că, arhitectura finală trebuie să fie una validă. Spre exemplu, pentru a adăuga primul nivel de convoluție la model, se selectează prin intermediul radio-buttonului nivelul Conv2D și se completează ca număr de canale de intrare 3 (o poză color are 3 canale și anume rgb) și un număr de canale de ieșire(câte filtre de convoluție vor fi aplicate), spre exemplu 32. După completare, se apasă butonul Add layer pentru a adăuga nivelul la model. Se procedează similar pentru celelalte nivele, până la completarea arhitecturii. În cazul în care se introduce accidental un nivel greșit se poate apăsa butonul Reset model și apoi procedura de inserție de nivele va trebui reluată de la început. La final, se apasă butonul Compile model ceea ce va începe etapele de antrenament și validare. Timpul de execuție al acestor etape depinde de dimensiunea datelor și de librăria aleasa. Pentru un set de date mic poate dura câteva minute, iar pentru seturi de date de milioane de imagini poate dura ore sau chiar zile. După ce etapa de antrenament și validare este încheiata, vor fi afișate metricile calculate pentru ultima epoca. Apoi modelul poate fi încărcat și se pot face precizii pe imagini noi.

7 BIBLIOGRAFIE

- [Cam01] [What is a tensor?](#)
- [Che21] [Chen James, Neural Network](#)
- [Cnn21] [ML Practicum: Image Classification](#)
- [IBM20] [Neural Networks](#)
- [Wik01] [Artificial neuron](#)
- [Wik02] [Artificial neural network](#)
- [Wik03] [Kaggle](#)
- [Wik04] [Tensor](#)

A. CODUL SURSĂ

main.py:

```
from gui import Gui

gui = Gui()
```

ModelInterface.py:

```
class ModelInterface(object):
    def __init__(self):
        pass

    def read_dataset(self):
        raise Exception("NotImplementedException")

    def augment_data(self):
        raise Exception("NotImplementedException")

    def compile_model(self):
        raise Exception("NotImplementedException")

    def train(self):
        raise Exception("NotImplementedException")

    def save_model(self):
        raise Exception("NotImplementedException")
```

CnnKeras.py:

```
import tensorflow as tf
from keras.applications.vgg16 import VGG16
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
```

```

import matplotlib.pyplot as plt
from tensorflow.keras.applications.resnet import ResNet50
from tensorflow.keras.models import Model
# from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Sequential
import visualkeras

from ModelInterface import ModelInterface

class CNNKeras(ModelInterface):
    def __init__(self, train_ds_path, val_ds_path, batch_size, image_size,
epochs, model_save_path, model):
        self.BATCH_SIZE = batch_size
        self.IMAGE_SIZE = image_size
        self.EPOCHS = epochs
        self.train_ds_path = train_ds_path
        self.val_ds_path = val_ds_path
        self.model_save_path = model_save_path
        self.train_dataset = 0
        self.val_dataset = 0
        self.train_dataset1 = 0
        self.val_dataset1 = 0
        self.class_names = 0
        self.data_augmentation = 0
        self.num_classes = 0
        self.model = model

    def read_dataset(self):
        self.train_dataset =
tf.keras.preprocessing.image_dataset_from_directory(
        self.train_ds_path,
        shuffle=True,
        image_size=(self.IMAGE_SIZE, self.IMAGE_SIZE),
        batch_size=self.BATCH_SIZE
    )
        self.val_dataset =
tf.keras.preprocessing.image_dataset_from_directory(
        self.val_ds_path,
        shuffle=True,
        image_size=(self.IMAGE_SIZE, self.IMAGE_SIZE),
        batch_size=self.BATCH_SIZE
    )
        self.class_names = self.train_dataset.class_names
        self.num_classes = len(self.class_names)
        # self.train_dataset =
self.train_dataset.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOT
UNE)
        # self.val_dataset =
self.val_dataset.cache().prefetch(buffer_size=tf.data.AUTOTUNE)

    def augment_data(self):
        self.data_augmentation = tf.keras.Sequential(
            [
layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical",
input_shape=(self.IMAGE_SIZE,

```

```

self.IMAGE_SIZE,
3)),
        layers.experimental.preprocessing.RandomRotation(0.1),
        layers.experimental.preprocessing.RandomZoom(0.1),
    ]
)

def compile_model(self):
    self.model.compile(optimizer=tf.keras.optimizers.Adam(),
loss=tf.keras.losses.BinaryCrossentropy(from_logits=False),
        metrics=['accuracy',
tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall'),
tf.keras.metrics.SpecificityAtSensitivity(0.5,
name='specificity_at_sensitivity')])
    self.model.summary()
    visulkeras.layered_view(self.model, legend=True,
to_file="output.png").show()

def train(self):
    history = self.model.fit(
        self.train_dataset,
        epochs=self.EPOCHS,
        batch_size=self.BATCH_SIZE,
        validation_data=self.val_dataset,
        verbose=1,
    )
    return history

def save_model(self):
    self.model.save(self.model_save_path)

```

ConvNeuralNet.py:

```

import torch.nn as nn

class ConvNeuralNet(nn.Module):
    def __init__(self, layers, linear_layers):
        super(ConvNeuralNet, self).__init__()
        self.layers = layers
        self.linear_layers = linear_layers

    # Progresses data across layers
    def forward(self, x):
        for value in self.layers:
            x = value(x)
        x = x.reshape(x.size(0), -1)
        for value in self.linear_layers:
            x = value(x)
        return x

```

CnnPytorch:

```

# Import Libraries
import time
import torch
import torch.nn as nn
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader
from torchvision import transforms
from ConvNeuralNet import ConvNeuralNet
from ConvNeuralNet3Layers import ConvNeuralNet3Layers
from ModelInterface import ModelInterface
from torchinfo import summary

class CNNPYtorch(ModelInterface):
    def __init__(self, train_ds_path, validation_ds_path, batch_size,
image_size, epochs, model_save_path
, model_layers, model_linear_layers
):
        self.BATCH_SIZE = batch_size
        self.IMAGE_SIZE = image_size
        self.EPOCHS = epochs
        self.train_ds_path = train_ds_path
        self.validation_ds_path = validation_ds_path
        self.model_save_path = model_save_path
        self.train_dataset = 0
        self.test_dataset = 0
        self.train_dl = 0
        self.test_dl = 0
        self.class_names = 0
        self.data_augmentation = 0
        self.num_classes = 2
        self.model = 0
        self.scores = 0
        self.device = "cpu"
        self.criterion = 0
        self.optimizer = 0
        self.TP = 0
        self.FP = 0
        self.TN = 0
        self.FN = 0
        self.val_TP = 0
        self.val_FP = 0
        self.val_TN = 0
        self.val_FN = 0
        self.val_history = 0
        self.model_layers = model_layers
        self.model_linear_layers = model_linear_layers
        self.history = [[0 for x in range(10)] for y in range(self.EPOCHS)]

    def read_dataset(self):
        train_transformation = transforms.Compose([
            transforms.Resize((self.IMAGE_SIZE, self.IMAGE_SIZE)),
            transforms.RandomHorizontalFlip(),
            transforms.RandomVerticalFlip(),
            transforms.RandomRotation(0.12),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.4914, 0.4822, 0.4465],
std=[0.2023, 0.1994, 0.2010])
])

```

```

        test_transformation = transforms.Compose([
            transforms.Resize((self.IMAGE_SIZE, self.IMAGE_SIZE)),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.4914, 0.4822, 0.4465],
std=[0.2023, 0.1994, 0.2010])
        ])
        # Load train, validation and test dataset
        train_file = self.train_ds_path
        test_file = self.validation_ds_path

        self.train_dataset = ImageFolder(train_file, train_transformation)
        self.test_dataset = ImageFolder(test_file, test_transformation)

        # PyTorch data loaders
        self.train_dl = DataLoader(self.train_dataset,
batch_size=self.BATCH_SIZE, shuffle=True)
        self.test_dl = DataLoader(self.test_dataset,
batch_size=self.BATCH_SIZE)
        self.class_names = self.train_dl.dataset.classes

        def compile_model(self):
            self.model = ConvNeuralNet(self.model_layers,
self.model_linear_layers)
            # self.model = ConvNeuralNet3Layers()
            # Set Loss function with criterion
            self.criterion = nn.BCELoss()
            # Set optimizer with optimizer
            self.optimizer = torch.optim.Adam(self.model.parameters(),
lr=0.001)

        def compute_measurements(self, loc_TP, loc_FP, loc_TN, loc_FN):
            # Acuratete = TP+TN/(TP+TN+FP+FN) -> toate corecte din toate
            # Precision = TP / (TP+FP) -> true positiv din toate pozitivele
prezise
            # Recall = TP / P -> true positiv din toate pozitivele reale
            # Sensitivity = Recall
            # Specificity = TN / (TN + FP) -> true negativ din toate negativele
reale
            accuracy = ((loc_TP + loc_TN) / (loc_TP + loc_TN + loc_FP +
loc_FN)) * 100
            if loc_TP + loc_FP > 0:
                precision = loc_TP / (loc_TP + loc_FP)
            else:
                precision = 0
            if (loc_TP + loc_FN) > 0:
                recall = loc_TP / (loc_TP + loc_FN)
            else:
                recall = 0
            if (loc_TN + loc_FP) > 0:
                specificity = loc_TN / (loc_TN + loc_FP)
            else:
                specificity = 0
            return accuracy, precision, recall, specificity

        def update_confusionmatrix(self, y_actual, y_hat, loc_TP, loc_FP,
loc_TN, loc_FN):
            # print(y_hat)
            # print(y_actual)
            for i in range(len(y_hat)):

```

```

        if y_actual[i].item() > 0.5 and y_hat[i].item() == 1:
            loc_TP += 1
        if y_hat[i].item() == 0 and y_actual[i].item() > 0.5:
            loc_FP += 1
        if y_actual[i].item() <= 0.5 and y_hat[i].item() == 0:
            loc_TN += 1
        if y_hat[i].item() == 1 and y_actual[i].item() <= 0.5:
            loc_FN += 1
    return loc_TP, loc_FP, loc_TN, loc_FN

def train(self):
    # PyTorch - Training the Model
    for e in range(self.EPOCHS):
        self.model.train()
        # define the loss value after the epoch
        losss = 0.0
        number_of_sub_epoch = 0
        start = time.time()
        # loop for every training batch (one epoch)
        for images, labels in self.train_dl:
            # in pytorch you have assign the zero for gradien in any
sub epoch
            self.optimizer.zero_grad()
            # create the output from the network
            out = self.model(images)
            _, predicted = out.max(1)
            # count the loss function
            loss = self.criterion(out, labels.unsqueeze(1).float())
            # count the backpropagation
            loss.backward()
            # learning
            self.optimizer.step()
            # add new value to the main loss
            losss += loss.item()
            number_of_sub_epoch += 1
            self.TP, self.FP, self.TN, self.FN =
self.update_confusionmatrix(out, labels, self.TP, self.FP, self.TN,
self.FN)

            end = time.time()
            print("Epoch {}: Loss: {} Time: {} ".format(e, losss /
number_of_sub_epoch, (end - start)))
            print("TP: {} FP: {} TN: {} FN: {}".format(self.TP, self.FP,
self.TN, self.FN))
            accuracy, precision, recall, specificity =
self.compute_measurements(self.TP, self.FP, self.TN, self.FN)

            print("Accuracy: {} Precision: {} Recall: {} Specificity: {}
".format(accuracy,
precision,
recall,
specificity))
            self.history[e][0] = accuracy
            self.history[e][1] = precision
            self.history[e][2] = recall
            self.history[e][3] = specificity

```

```

        self.history[e][4] = losss / number_of_sub_epoch

        self.TP = 0
        self.FP = 0
        self.TN = 0
        self.FN = 0
        self.validate(e)

    return self.history

def save_model(self):
    torch.save(self.model, self.model_save_path)

def load_model(self):
    self.model =
torch.load(('E:/automatica/master_an2/Disertatie/Breast_xray/breast_pytorch
_CNN.pth'))
    self.criterion = nn.BCELoss()
    # Set optimizer with optimizer
    self.optimizer = torch.optim.Adam(self.model.parameters(),
lr=0.001)

def validate(self, e):
    # PyTorch - Comparing the Results
    total = 0
    val_loss = 0.0
    val_lossss = 0.0
    number_of_sub_epoch = 0
    self.model.eval()
    # self.optimizer.zero_grad()
    with torch.no_grad():
        for images, labels in self.test_dl:
            outputs = self.model(images)
            _, predicted = outputs.max(1)
            val_loss = self.criterion(outputs,
labels.unsqueeze(1).float())
            number_of_sub_epoch += 1
            total += labels.size(0)
            val_lossss += val_loss.item()
            # print(outputs)
            self.val_TP, self.val_FP, self.val_TN, self.val_FN =
self.update_confusionmatrix(outputs, labels,

self.val_TP,

self.val_FP,

self.val_TN,

self.val_FN)
            accuracy, precision, recall, specificity =
self.compute_measurements(self.val_TP, self.val_FP, self.val_TN,

self.val_FN)
            print('On {} test images: Val_Loss {}% with PyTorch'.format(total,
val_lossss / number_of_sub_epoch))
            print("Validation TP: {} FP: {} TN: {} FN: {}".format(self.val_TP,
self.val_FP, self.val_TN, self.val_FN))
            print("Val_Accuracy: {} Val_Precision: {} Val_Recall: {}".format(

```



```

Val_Specificity: {} ".format(accuracy,
precision,
recall,
specificity))
    self.history[e][5] = accuracy
    self.history[e][6] = precision
    self.history[e][7] = recall
    self.history[e][8] = specificity
    self.history[e][9] = val_loss / number_of_sub_epoch

    self.val_TP = 0
    self.val_FP = 0
    self.val_TN = 0
    self.val_FN = 0

```

Gui.py:

```

import os
from tkinter import filedialog
from tkinter import *
import tkinter as tk

import numpy
from PIL import ImageTk, Image
import torch
from keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Rescaling, Flatten
from keras.models import load_model, Sequential
from matplotlib import pyplot as plt
from torchvision.transforms import transforms
import tensorflow as tf

from CNNKeras import CNNKeras
from CNNPytorch import CNNPytorch
from ConvNeuralNet import ConvNeuralNet

class Gui:
    def __init__(self):
        self.classes = {0: 'normal', 1: 'cancer'}
        self.top = tk.Tk()
        canvas = Canvas(
            self.top,
            height=1024,
            width=1024,
            bg="#fff"
        )
        canvas.place(x=500, y=0)

        self.backgroundcolor = "#3399ff"
        canvas.create_rectangle(
            0, 0, 1024, 1024,
            outline="#fb0",
            fill=self.backgroundcolor)
        self.label = Label(self.top, font=('arial', 15, 'bold'))
        self.labelAccuracy = Label(self.top, font=('arial', 15, 'bold'))
        self.radioVar = StringVar(self.top, "1")

```

```

self.breast_image = Label(self.top)
self.filepath = ""
self.EPHOCS = 5
self.ephocs_textbox = Entry(self.top)
self.batch_size = 64
self.batch_size_textbox = Entry(self.top)
self.number_layers = 10
self.number_layers_textbox = Entry(self.top)
self.cnn_current_layer = ""
self.cnn_current_layer_textbox = Entry(self.top)
self.radiovar_cnn_current_layer = StringVar(self.top, "1")
self.cnn_in_channel = Entry(self.top)
self.cnn_out_channel = Entry(self.top)
self.cnn_kernel_size = Entry(self.top)
self.maxpool_kernel_size = Entry(self.top)
self.maxpool_stride = Entry(self.top)
self.dropout = Entry(self.top)
self.linear_in_channel = Entry(self.top)
self.linear_out_channel = Entry(self.top)
self.model_keras = Sequential()
self.model_keras.add(Rescaling(1. / 255, input_shape=(50, 50, 3)))
self.model_pytorch_layers = torch.nn.ModuleList()
self.model_pytorch_linear_layers = torch.nn.ModuleList()
self.train_ds_path = Entry(self.top)
self.val_ds_path = Entry(self.top)
self.save_path = Entry(self.top)
self.labelResults = Label(self.top, font=('arial', 10, 'bold'))
self.labelResults.place(x=515,y=415)
self.init_gui()

# initialize GUI
def init_gui(self):
    self.top.geometry('1024x1024')
    self.top.title('Image classification')
    self.top.configure(background='#FFFFFF')
    upload = Button(self.top, text="Upload an image",
command=self.upload_image, padx=10, pady=5)
    # Dictionary to create multiple buttons
    values = {"Pytorch": "1",
              "Keras": "2"
            }
    upload.configure(background='#364156', foreground='white',
font=('arial', 10, 'bold'))
    upload.place(x=100, y=500)
    self.breast_image.place(x=20, y=200)
    self.label.place(x=100, y=150)
    self.labelAccuracy.place(x=200, y=150)
    # Loop is used to create multiple Radio buttons rather than
    creating each button separately
    i = 0
    for (text, value) in values.items():
        i += 1
        Radiobutton(self.top, text=text, font=('arial', 10, 'bold'),
variable=self.radioVar, background='#FFFFFF',
value=value).place(x=100, y=50 + 25 * i)
    tk.Button(self.top, text="Load model", command=self.open_file,
background='#364156', foreground='white', font=('arial', 10,
'bold')).place(x=100, y=125)
    Label(self.top, text="Number of epochs", font=('arial', 10,

```

```

'bold'), foreground='white', background=self.backgroundcolor).place(x=550,
y=75)
    self.epochs_textbox.place(x=550, y=110)
    Label(self.top, text="Batch size", font=('arial', 10, 'bold'),
foreground='white', background=self.backgroundcolor).place(x=550, y=145)
    self.batch_size_textbox.place(x=550, y=180)
    Label(self.top, text="Training dataset path", font=('arial', 10,
'bold'), foreground='white', background=self.backgroundcolor).place(x=550,
y=205)
    self.train_ds_path.place(x=550, y=240)
    Label(self.top, text="Validation dataset path", font=('arial', 10,
'bold'), foreground='white', background=self.backgroundcolor).place(x=550,
y=275)
    self.val_ds_path.place(x=550, y=310)
    Label(self.top, text="Save model path", font=('arial', 10, 'bold'),
foreground='white', background=self.backgroundcolor).place(x=550, y=345)
    self.save_path.place(x=550, y=380)

    Radiobutton(self.top, text="Conv2D", font=('arial', 10, 'bold'),
variable=self.radiovar_cnn_current_layer, background=self.backgroundcolor,
value="1").place(x=800, y=75)
    Label(self.top, text="In channels (only pytorch)", font=('arial',
10, 'bold'), foreground='white',
background=self.backgroundcolor).place(x=800, y=100)
    self.cnn_in_channel.place(x=800, y=125)
    Label(self.top, text="Out channels", font=('arial', 10, 'bold'),
foreground='white', background=self.backgroundcolor).place(x=800, y=150)
    self.cnn_out_channel.place(x=800, y=175)
    Label(self.top, text="Kernel size", font=('arial', 10, 'bold'),
foreground='white', background=self.backgroundcolor).place(x=800, y=200)
    self.cnn_kernel_size.place(x=800, y=225)
    Radiobutton(self.top, text="MaxPool2d", font=('arial', 10, 'bold'),
background=self.backgroundcolor, variable=self.radiovar_cnn_current_layer,
value="2").place(x=800, y=250)
    Label(self.top, text="Kernel size (only pytorch)", font=('arial',
10, 'bold'), foreground='white',
background=self.backgroundcolor).place(x=800, y=275)
    self.maxpool_kernel_size.place(x=800, y=300)
    Label(self.top, text="Stride (only pytorch)", font=('arial', 10,
'bold'), foreground='white', background=self.backgroundcolor).place(x=800,
y=325)
    self.maxpool_stride.place(x=800, y=350)
    Radiobutton(self.top, text="Dropout", font=('arial', 10, 'bold'),
variable=self.radiovar_cnn_current_layer, background=self.backgroundcolor,
value="3").place(x=800, y=375)
    Label(self.top, text="Dropout rate", font=('arial', 10, 'bold'),
foreground='white', background=self.backgroundcolor).place(x=800, y=400)
    self.dropout.place(x=800, y=425)
    Radiobutton(self.top, text="Linear", font=('arial', 10, 'bold'),
variable=self.radiovar_cnn_current_layer, background=self.backgroundcolor,
value="4").place(x=800, y=450)
    Label(self.top, text="In channels (only pytorch)", font=('arial',
10, 'bold'), foreground='white',
background=self.backgroundcolor).place(x=800, y=475)
    self.linear_in_channel.place(x=800, y=500)
    Label(self.top, text="Out channels", font=('arial', 10, 'bold'),
foreground='white', background=self.backgroundcolor).place(x=800, y=525)
    self.linear_out_channel.place(x=800, y=550)
    Radiobutton(self.top, text="Sigmoid", font=('arial', 10, 'bold'),

```

```

variable=self.radiovar_cnn_current_layer, background=self.backgroundcolor ,
        value="5").place(x=800, y=575)
        tk.Button(self.top, text="Add layer", command=self.add_layer,
background='#364156', foreground='white', font=('arial', 10,
'bold')).place(x=800, y=600)
        tk.Button(self.top, text="Compile model",
command=self.compile_model, background='#364156', foreground='white',
font=('arial', 10, 'bold')).place(x=800, y=650)
        tk.Button(self.top, text="Reset model", command=self.reset_model,
background='#364156', foreground='white', font=('arial', 10,
'bold')).place(x=800, y=700)

        self.top.mainloop()

    def reset_model(self):
        self.model_keras = Sequential()
        self.model_keras.add(Rescaling(1. / 255, input_shape=(50, 50, 3)))
        self.model_pytorch_layers = torch.nn.ModuleList()
        self.model_pytorch_linear_layers = torch.nn.ModuleList()
        self.labelResults.configure(foreground='#ffffff',
background=self.backgroundcolor, text="")

    def add_layer(self):
        layer_type = str(self.radiovar_cnn_current_layer.get())
        if "2" == str(self.radioVar.get()):
            if "1" == layer_type:

self.model_keras.add(Conv2D(int(self.cnn_out_channel.get()),
int(self.cnn_kernel_size.get()),
                                padding='same',
activation='relu'))
            if "2" == layer_type:
                self.model_keras.add(MaxPooling2D())
            if "3" == layer_type:
                self.model_keras.add(Dropout(float(self.dropout.get())))
                self.model_keras.add(Flatten())
            if "4" == layer_type:

self.model_keras.add(Dense(int(self.linear_out_channel.get()),
activation='relu'))
            if "5" == layer_type:
                self.model_keras.add(Dense(1, activation='sigmoid'))
            elif "1" == str(self.radioVar.get()):
                if "1" == layer_type:

self.model_pytorch_layers.append(torch.nn.Conv2d(in_channels=int(self.cnn_i
n_channel.get()), out_channels=int(self.cnn_out_channel.get()),
kernel_size=int(self.cnn_kernel_size.get()))
                self.model_pytorch_layers.append(torch.nn.ReLU())
            if "2" == layer_type:

self.model_pytorch_layers.append(torch.nn.MaxPool2d(int(self.maxpool_kernel
_size.get()), int(self.maxpool_stride.get())))
            if "3" == layer_type:

self.model_pytorch_layers.append(torch.nn.Dropout(float(self.dropout.get())
))
            if "4" == layer_type:

```

```

self.model_pytorch_linear_layers.append(torch.nn.Linear(int(self.linear_in_
channel.get()), int(self.linear_out_channel.get())))
    if "5" == layer_type:
        self.model_pytorch_linear_layers.append(torch.nn.Sigmoid())

def compile_model(self):
    self.EPHOCS = int(self.ephocs_textbox.get())
    self.batch_size = int(self.batch_size_textbox.get())
    self.labelResults.configure(foreground='#ffffff',
background=self.backgroundcolor, text="")
    if "2" == str(self.radioVar.get()):
        algorithm_cnn = CNNKeras(self.train_ds_path.get(),
                                self.val_ds_path.get(),
                                self.batch_size,
                                50,
                                self.EPHOCS,
                                self.save_path.get(),
                                self.model_keras
                                )
        algorithm_cnn.read_dataset()
        algorithm_cnn.augment_data()
        algorithm_cnn.compile_model()
        history = algorithm_cnn.train()
        self.labelResults.configure(foreground='#ffffff',
                                background=self.backgroundcolor,
                                text="Precision = {:.2f}\n Accuracy
= {:.2f}\n Recall = {:.2f}\n"
                                "Specificity = {:.2f}\n Loss =
{: .2f}\n" "Validation Precision = {:.2f}\n"
                                "Validation Accuracy =
{: .2f}\n Validation Recall = {:.2f}\n"
                                "Validation Specificity =
{: .2f}\n"
                                "Validation Loss =
{: .2f}\n".format(float(history.history["precision"][0]),
float(history.history["accuracy"][0]),
float(history.history["recall"][0]),
float(history.history["specificity_at_sensitivity"][0]),
float(history.history["loss"][0]),
float(history.history["val_precision"][0]),
float(history.history["val_accuracy"][0]),
float(history.history["val_recall"][0]),
float(history.history[
"val_specificity_at_sensitivity"][
0]),

```

```

float(history.history["val_loss"][0]),
)

        )

        algorithm_cnn.save_model()
    elif "1" == str(self.radioVar.get()):
        algorithm_cnn = CNNPytorch(self.train_ds_path.get(),
                                    self.val_ds_path.get(),
                                    self.batch_size,
                                    50,
                                    self.EPHOCS,
                                    self.save_path.get(),
                                    self.model_pytorch_layers,
                                    self.model_pytorch_linear_layers
                                    )
        algorithm_cnn.read_dataset()
        algorithm_cnn.compile_model()
        history = algorithm_cnn.train()
        self.labelResults.configure(foreground='#ffffff',
                                    background=self.backgroundcolor,
                                    text="Precision = {:.2f}\n Accuracy
= {:.2f}\nRecall = {:.2f}\n"
                                    "Specificity = {:.2f}\nLoss =
{: .2f}\n""Validation Precision = {:.2f}\n"
                                    "Validation Accuracy =
{: .2f}\nValidation Recall = {:.2f}\n"
                                    "Validation Specificity =
{: .2f}\n"
                                    "Validation Loss =
{: .2f}\n".format(float(history[self.EPHOCS-1][0]),
float(history[self.EPHOCS-1][1]),
float(history[self.EPHOCS-1][2]),
float(history[self.EPHOCS-1][3]),
float(history[self.EPHOCS-1][4]),
float(history[self.EPHOCS-1][5]),
float(history[self.EPHOCS-1][6]),
float(history[self.EPHOCS-1][7]),
float(history[self.EPHOCS-1][8]),
float(history[self.EPHOCS-1][9])
)

        )

        algorithm_cnn.save_model()

    def open_file(self):
        file = filedialog.askopenfile(mode='r')
        if file:
            self.filepath = os.path.abspath(file.name)

    def classify(self, file_path):

```

```

        if "1" == str(self.radioVar.get()):
            #
            'E:/automatica/master_an2/Disertatie/Breast_xray/breast_pytorch_CNN.pth'
            model1 = torch.load(self.filepath)
            model1.eval()
            normalize = transforms.Normalize(
                mean=[0.4914, 0.4822, 0.4465], std=[0.2023, 0.1994, 0.2010]
            )
            preprocess = transforms.Compose([
                transforms.Resize(50),
                transforms.ToTensor(),
                normalize
            ])
            img_pil = Image.open(file_path)
            img_tensor = preprocess(img_pil).float()
            img_tensor = img_tensor.unsqueeze_(0)
            fc_out = model1(img_tensor)
            output = fc_out.detach().numpy()
            if 0.5 > float(output[0]):
                sign = 0
                self.label.configure(foreground='#011638',
background="#ffffff", text=self.classes[sign])
                self.labelAccuracy.configure(foreground='#011638',
background="#ffffff", text=1 - float(output[0]))
            else:
                sign = 1
                self.label.configure(foreground='#011638',
background="#ffffff", text=self.classes[sign])
                self.labelAccuracy.configure(foreground='#011638',
background="#ffffff", text=float(output[0]))

        elif "2" == str(self.radioVar.get()):
            # 'breast_keras_CNN.h5'
            model = load_model(self.filepath)
            img = tf.keras.utils.load_img(
                file_path, target_size=(50, 50)
            )
            # Create a batch
            img_array = tf.keras.utils.img_to_array(img)
            img_array = tf.expand_dims(img_array, 0)
            # classify image
            predictions = model.predict(img_array)
            if 0.5 > predictions[0]:
                sign = self.classes[0]
                self.labelAccuracy.configure(foreground='#011638',
background="#ffffff", text=1 - float(predictions[0]))
                self.label.configure(foreground='#011638',
background="#ffffff", text=sign)
            else:
                sign = self.classes[1]
                self.labelAccuracy.configure(foreground='#011638',
background="#ffffff", text=float(predictions[0]))
                self.label.configure(foreground='#011638',
background="#ffffff", text=sign)
            else:
                self.label.configure(foreground='#011638',
background="#ffffff", text="no prediction possible")
            print(self.radioVar)

```

```

def show_classify_button(self, file_path):
    classify_b = Button(self.top, text="Classify Image",
command=lambda: self.classify(file_path), padx=10, pady=5)
    classify_b.configure(background='#364156', foreground='white',
font=('arial', 10, 'bold'))
    classify_b.place(x=100, y=550)

def upload_image(self):
    try:
        file_path = filedialog.askopenfilename()
        uploaded = Image.open(file_path)
        resize_image = uploaded.resize((250, 250))
        uploaded.thumbnail(((self.top.wininfo_width() / 2.25),
(self.top.wininfo_height() / 2.25)))
        im = ImageTk.PhotoImage(resize_image)
        self.breast_image.configure(image=im, height=250, width=250)
        self.breast_image.image = im
        self.label.configure(text='')
        self.labelAccuracy.configure(text='')
        self.show_classify_button(file_path)
    except:
        pass

```