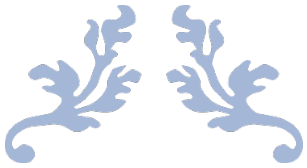

BRAIN ANOMALY DETECTION

Detect anomalies in CT scans of the brain

APRIL 14, 2023

Mihailescu Teodor
Grupa 242

Contents

Mihailescu Teodor
Grupa 242

I. Descriere proiect

Scopul este de a diferenția între două categorii de scanări CT ale creierului - cele care prezintă anomalii (etichetate cu 1) și cele considerate normale (etichetate cu 0).
Fiecare instanță din setul de date este o imagine alb-negru cu o rezoluție de 224x224 pixeli.
Fiecare instanță este încadrată într-una dintre cele două clase, cu 15.000 de exemple etichetate în setul de antrenare, 2.000 de exemple etichetate în setul de validare și încă 5.149 de exemple în setul de testare. Etichetele pentru setul de testare nu sunt incluse în setul de date.

II. Set de date

- data.zip: conține mostre de imagini în format .PNG, cu o probă per fișier
- train_labels.txt: furnizează etichetele de antrenament cu o etichetă per rând
- validation_labels.txt: furnizează etichetele de validare cu o etichetă per rând
- sample_submission.csv: un fișier cu o probă de trimitere în formatul correct

Fișierul train_labels_.txt conține etichetele asociate cu mostrele de antrenament. Fiecare etichetă este furnizată într-un rând separat, cu un format specific.

id,class

000001,0

...

017000,1

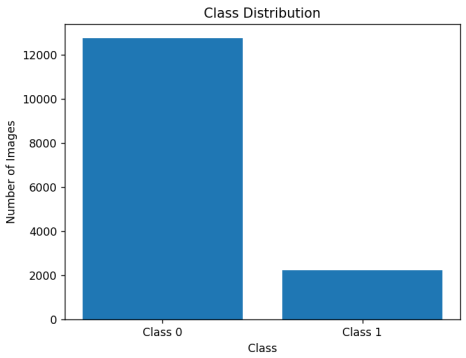

Fig.1 Train Labels Distribution

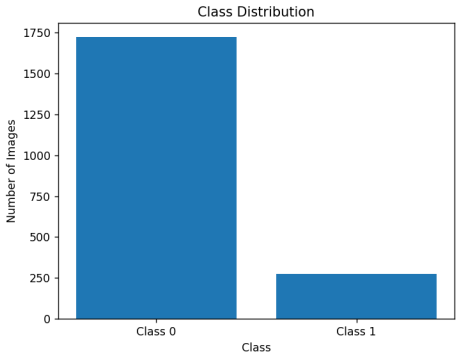

Fig.2 Validation Labels Distribution

Mihailescu Teodor
Grupa 242

III. Abordari

A. Decision Tree Classifier

Ca prima abordare am ales sa folosesc un clasificator de arbore de decizie pentru a clasifica imaginile creierului ca normale sau anormale pe baza valorilor lor medii și a intensității abaterii standard. Un clasificator cu arbori de decizie este un model de învățare automată folosit pentru a clasifica un set de date în funcție de anumite caracteristici. Un arbore de decizie este un model în formă de arbore care ajută la luarea deciziilor și clasificarea datelor. Fiecare nod al arborelui reprezintă o caracteristică și fiecare ramură a nodului reprezintă o alegere posibilă pentru acea caracteristică. Modelul începe de la rădăcină și navighează prin ramuri până când ajunge la o frunză, care reprezintă o clasificare a datelor de intrare. Clasificarea se face în funcție de valorile caracteristicilor de intrare și a regulilor învățate din datele de antrenare.

Pornind de la acest prim model, am folosit o citire in care încarc datele(imaginile), iar pentru fiecare imagine o transform într-o matrice, după care calculez media și deviația standard a intensității pixelilor și se adaugă aceste valori la datele de antrenare. Pentru a citi datele din folder-ul unibuc-brain-ad am folosit librăria PIL, totodată, calcuând si valorea medie si intensității abaterii standard. Am utilizat np array-uri care cuprind:

img_arr: Acesta este un array numpy care conține reprezentarea numerică a datelor de imagine. În mod specific, acesta conține intensitățile pixelilor imaginii în tonuri de gri sub formă de un array 2D. Dimensiunile acestui array sunt (244 x 224), unde 244 reprezintă înălțimea și 224 reprezintă lățimea imaginii. Fiecare element al array-ului reprezintă intensitatea pixelului la acea locație în imagine. Valorile din acest array variază între 0 și 255, unde 0 reprezintă negru și 255 reprezintă alb.

train_data: Acest array stochează valorile calculate ale mediei și deviației standard pentru fiecare imagine. Mai precis, conține un array 2D în care fiecare rând reprezintă o imagine, iar fiecare coloană reprezintă o caracteristică (adică valoarea medie sau deviația standard). Dimensiunile acestui array sunt (numărul de imagini x 2).

Ambele array-uri numpy conțin valori numerice care vor fi utilizate pentru a antrena un model de învățare automată. Array-ul img_arr conține datele reale ale imaginii în formă numerică, în timp ce array-ul train_data conține statistici sumare (media și deviația standard) care sunt calculate din datele imaginii.

Pentru a evita discrepanța dintre cele două clase folosim un dictionar (o colecție de tip cheie unica valoare) unde calculăm prioritățile de clasă și aplicăm ponderi ale clasei fiecărui eșantion din datele de antrenament pe baza etichetei sale.

În cele din urmă, sunt calculate ponderile de eșantion pentru fiecare eșantion de antrenament în funcție de eticheta sa de clasă. Ponderile de eșantion sunt calculate ca

raportul dintre ponderea clasei și probabilitatea de clasă. Acest lucru înseamnă că eșantioanele din clasele rare(clasa 1) vor avea ponderi mai mari, ceea ce ajută la echilibrarea procesului de antrenare și previne modelul să fie afectat de clasa majoritară. După ce ponderile de eșantion sunt calculate, este creat și antrenat un model DecisionTreeClassifier folosind metoda fit, unde parametrul sample_weight este setat la ponderile de eșantion calculate. Acest lucru asigură faptul că modelul ia în considerare dezechilibrul de clasă în luarea deciziilor.

Pentru acest model am obținut 0.7825 acuratețe pe validation_data, iar pe sample_submission scorul: 0.23539. Din păcate acest model nu a fost destul de eficient, așadar am hotărât un model nou care care augmentează setul de date corespunzător și preprocesează imaginile mai bine.

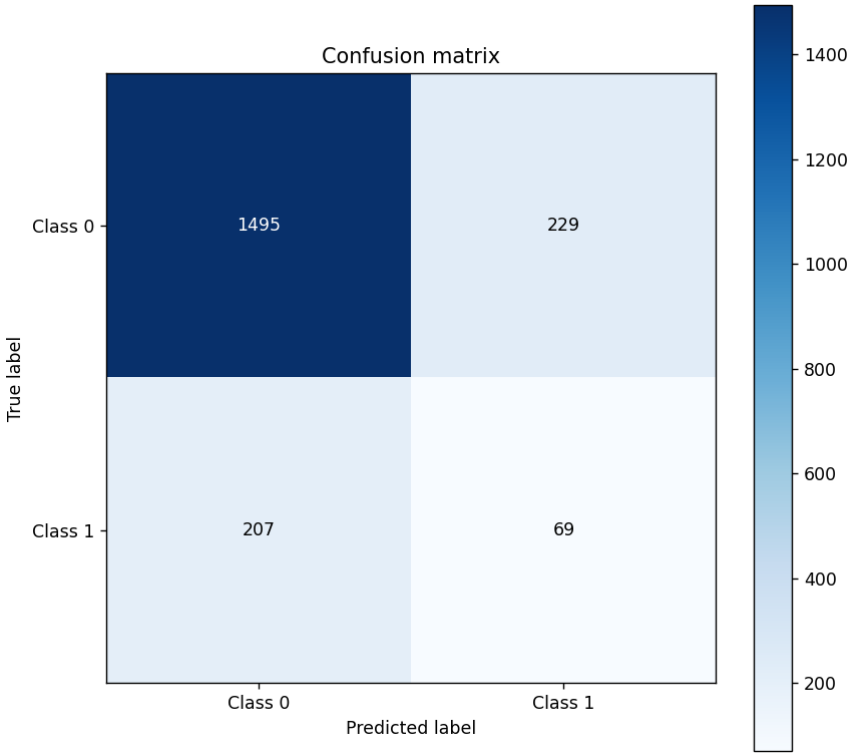

Fig. 3 Matricea de confuzie pentru Modelul Decision Tree Classifier pe validation_labels

	precision	recall	f1-score	support
0	0.88	0.87	0.87	1724
1	0.23	0.25	0.24	276
accuracy			0.78	2000
macro avg	0.55	0.56	0.56	2000
weighted avg	0.79	0.78	0.79	2000

Fig. 4 raportul ce cuprinde accuracy rates, precision and recall for each class

B. CNN

CNN este prescurtarea de la Convolutional Neural Network și este o rețea neuronală utilizată pentru recunoașterea imaginilor, clasificare și procesare de imagini. Această rețea este formată din straturi de convoluție, poolare și complet conectate. Straturile de convoluție folosesc filtre pentru a detecta caracteristici în imaginea de intrare, iar straturile de poolare reduc dimensiunea hărților de caracteristici și previn overfitting-ul. Straturile complet conectate se află la sfârșitul rețelei și iau decizia finală de clasificare.
Ca să fie algoritmul eficient verificăm dupa importul librăriilor dacă sunt disponibile dispozitive GPU, apoi setăm ca acestea să fie vizibile, totodată, alocând memorie pentru primul GPU din lista de dispozitive. Funcția set_memory_growth activează alocarea dinamică de memorie, ceea ce înseamnă că TensorFlow va aloca memorie pentru model pe măsură ce este necesar în timpul antrenării.

1. Ca și prim pas am citit imaginile, menționând color_mode="grayscale", rescalându-le la o dimensiunea de 150x150 de pixeli pentru a optimizia procesarea și memoria. Imaginea in tonuri de gri este apoi convertita la un array numpy folosind functia img_to_array din acelasi modul. Aceasta creeaza un array numpy 2D cu forma (inaltime, latime, 1).

2. Urmează procesarea și augmentarea datelor ce permit modelului să se antreneze pe o varietate mai largă de imagini și reducând riscul de supradimensionare. Clasa ImageDataGenerator primește un set de parametri care definesc cum să augmentăm datele de antrenament. În acest caz, sunt setați următorii parametri de augmentare:

Se creează doi generatori folosind clasa ImageDataGenerator. Train_generator este utilizat pentru a genera loturi de date de antrenament, iar val_generator este utilizat pentru a genera loturi de date de validare. Obiectul datagen este o instanță a clasei ImageDataGenerator care este utilizat pentru a defini parametrii de augmentare, cum ar fi intervalul de rotație, intervalul de schimbare a lățimii și înălțimii și intervalul de zoom.

Metoda flow() a clasei ImageDataGenerator generează loturi de date augmentate prin aplicarea parametrilor de augmentare la datele originale. Datele de antrenament și etichetele de antrenament sunt trecute ca intrare la metoda flow(), împreună cu batch_size, subset, care specifică dacă datele sunt pentru antrenament sau validare și shuffle, care specifică dacă datele ar trebui să fie amestecate după fiecare epocă.

- **rotation_range=10** : specifică intervalul de rotații aleatoare de aplicat imaginilor, în grade.
- **width_shift_range=0,1** : specifică intervalul de deplasări orizontale aleatoare de aplicat imaginilor, ca fracțiune din lățimea totală.
- **height_shift_range=0,1** : specifică intervalul de deplasări verticale aleatoare de aplicat imaginilor, ca fracțiune din înălțimea totală.
- **zoom_range=0,1** : specifică intervalul de zoom-uri aleatoare de aplicat imaginilor, ca fracțiune din dimensiunea originală.
- **horizontal_flip=True** : specifică dacă să se inverseze aleatoriu imaginile orizontal în timpul antrenamentului.
- **vertical_flip= True** : specifică dacă să se inverseze aleatoriu imaginile vertical în timpul antrenamentului.
- **validation_split= True** : specifică fracțiunea de date de antrenament de utilizat pentru validare.

3. Definirea modelului
    - Conv2D: Aceasta este o placă de convoluție 2D care aplică un set de filtre imaginii de intrare, glisându-le peste imagine pentru a produce un set de hărți de caracteristici. În acest caz, stratul are 32 de filtre, fiecare de dimensiune (3,3), care sunt aplicate cu o funcție de activare liniară rectificată (*relu*). Parametrul de umplere este setat la *same*, ceea ce înseamnă că intrarea este umplută cu zero, astfel încât ieșirea să aibă aceeași formă ca și intrarea. Forma de intrare este (150, 150, 1), ceea ce înseamnă că intrarea este o imagine cu o înălțime și lățime de 150 de pixeli și un canal de culoare.

- BatchNormalization: Acesta este un strat care normalizează ieșirile stratului anterior prin scăderea mediei și împărțirea la deviația standard.

- MaxPooling2D: Acesta este un strat care reduce dimensiunea intrării prin luarea valorii maxime în fiecare regiune (2,2) a intrării. Acest lucru reduce dimensiunea hărților de caracteristici și ajută la extragerea celor mai importante caracteristici din intrare.

- Flatten: Acesta este un strat care aplatizează ieșirea stratului anterior într-un vector 1D. Acest lucru este adesea utilizat pentru a trece de la straturi de convoluție la straturi complet conectate.

- Dense: Acesta este un strat complet conectat care aplică un set de greutăți la intrare și adaugă un termen de polarizare. Primul strat Dense are 128 de unități cu o funcție de activare *relu*, în timp ce al doilea strat Dense are o singură unitate de ieșire cu o funcție de activare *sigmoid*.

- Dropout: Acesta este o tehnică de regularizare care setează în mod aleatoriu o fracțiune din unitățile de intrare la zero în timpul antrenamentului. Acest lucru ajută la prevenirea overfitting-ului prin reducerea dependenței modelului de orice caracteristică.

4. Hiperparametrii:

   a. Straturi de convoluție:
      - Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(150, 150, 1))
      - Conv2D(64, (3, 3), activation='relu', padding='same')
      - Conv2D(128, (3, 3), activation='relu', padding='same')
      - Conv2D(256, (3, 3), activation='relu', padding='same')

      Aceste straturi efectuează convoluția imaginilor de intrare cu un set de filtre. Fiecare filtru învață să detecteze caracteristici specifice în imaginile de intrare. Funcția de activare utilizată este *relu*, care introduce non-liniaritate și ajută la învățarea de modele mai complexe. *padding='same'* asigură că hărțile de caracteristici de ieșire au aceeași dimensiune ca hărțile de caracteristici de intrare.

   b. Straturi de normalizare:
      BatchNormalization()

Normalizarea pe lot este utilizată pentru a normaliza activările straturilor de convoluție anterioare. Aceasta accelerează antrenarea, ajută la regularizare și reduce problema de deplasare a covariatei interne.

c. Straturi de reducere a dimensiunii:
MaxPooling2D((2, 2))
Redimensionarea maximă este utilizată pentru a micșora hărțile de caracteristici de ieșire din straturile de convoluție anterioare. Reduce dimensiunile spațiale ale ieșirii și ajută la extragerea celor mai relevante caracteristici, în timp ce ignoră cele irelevante.

d. Stratul de aplatizare:
Flatten()
Stratul de aplatizare este utilizat pentru a aplatiza ieșirile din straturile de convoluție și de reducere a dimensiunii anterioare într-un vector unidimensional, care poate fi utilizat ca intrare pentru straturile Dense.

e. Straturi Dense:
Dense(128, activation='relu')
Dense(1, activation='sigmoid')
Straturile Dense sunt straturi complet conectate care calculează ieșirea pe baza intrării din stratul anterior. Primul strat Dense are 128 de unități cu activare *relu* și este urmat de un strat BatchNormalization(). Al doilea strat Dense are o singură unitate cu activare *sigmoid* și produce ieșirea de clasificare binară.

f. Strat de abandonare:
Dropout(0.5)
Dropout este utilizat pentru a reduce suprapunerea prin abandonarea aleatorie a unor unități în stratul Dense în timpul antrenării. Ajută la prevenirea dependenței excesive a rețelei de un set mic de caracteristici și încurajează rețeaua să învețe caracteristici mai robuste.

5. Compilarea modelului

a. Parametrul *optimizer* specifică algoritmul de optimizare care va fi folosit pentru a minimiza funcția de pierdere. În acest caz, se folosește algoritmul Adam(Adaptive Moment Estimation) cu o rată de învățare (learning rate) de 0,001. Adam combină ideile din două algoritmi de optimizare - AdaGrad și RMSProp - și își ajustează rata de învățare în timpul antrenării în funcție de momentul gradului și de media pătratică

a gradientului. Astfel, Adam este mai rapid și mai eficient în comparație cu alte algoritmi de optimizare și poate ajuta la găsirea unui minim local al funcției de pierdere într-un timp mai scurt.

b. Parametrul *loss* specifică funcția de pierdere care va fi folosită în timpul antrenării. În acest caz, este utilizată funcția de pierdere "binary_crossentropy", care este folosită pentru problemele de clasificare binară. Scopul acestei funcții de pierdere este de a măsura cât de aproape sunt predicțiile de adevărata clasa. Cu cât valoarea acestei funcții de pierdere este mai mică, cu atât mai bine se potrivesc predicțiile modelului cu etichetele de clasă.

c. Parametrul *metrics* specifică metricile care vor fi utilizate pentru a evalua performanța modelului în timpul antrenării și testării. În acest caz, se utilizează metrica *accuracy*, care calculează proporția de exemple clasificate corect din totalul de exemple.

6. Model fit:
   Metoda fit() a modelului este apoi utilizată pentru a antrena modelul folosind loturile de date generate. Train_generator este trecut ca intrare pentru datele de antrenament, iar val_generator este trecut ca intrare pentru datele de validare. Parametrul **epochs=15** specifică de câte ori întregul set de date este antrenat. Parametrul **batch_size=32** specifică numărul de eșantioane per actualizare de gradient. Parametrul **steps_per_epoch= len(train_generator),** specifică numărul de loturi de eșantioane de utilizat în fiecare epocă. Parametrul **validation_steps= len(val_generator)** specifică numărul de loturi de eșantioane de validare de utilizat în fiecare epocă. Parametrul **class_weight- class_weights** este utilizat pentru a da greutate claselor în funcția de pierdere pentru a gestiona seturile de date neechilibrate.

7. Variante de CNN:
   Ca prima varianta am propus un model cu 12 layers, dar cu o singură BatchNormalization și cu un Dropout de 25%, alături de 10 epoci ceea ce a condus la un scor de 0.55497.

Layer Name	**Hyperparameters**
Conv2D_1	filters=32, kernel_size=(3, 3), activation='relu'
MaxPooling2D_1	pool_size=(2, 2)
Conv2D_2	filters=64, kernel_size=(3, 3), activation='relu'
MaxPooling2D_2	pool_size=(2, 2)

Mihailescu Teodor
Grupa 242

Layer Name	Hyperparameters
Conv2D_3	filters=128, kernel_size=(3, 3), activation='relu'
BatchNormalization	
MaxPooling2D_3	pool_size=(2, 2)
Flatten	
Dense_1	units=128, activation='relu'
BatchNormalization	
Dropout	rate=0.25
Dense_2	units=1, activation='sigmoid'

Apoi, am încercat să plec de la filters=64, batch_size=64 si epochs=10 ceea ce a dus la un scor de 0.56826. Am păstrat numărul de filters=64, batch_size-ul=32 si epochs=15 și am obținut 0.58028.

Layer Name	Hyperparameters
Conv2D_1	filters=64, kernel_size=(3, 3), activation='relu'
MaxPooling2D_1	pool_size=(2, 2)
Conv2D_2	filters=128, kernel_size=(3, 3), activation='relu'
MaxPooling2D_2	pool_size=(2, 2)
Conv2D_3	filters=256, kernel_size=(3, 3), activation='relu'
BatchNormalization	
MaxPooling2D_3	pool_size=(2, 2)
Flatten	
Dense_1	units=128, activation='relu'
BatchNormalization	
Dropout	rate=0.25
Dense_2	units=1, activation='sigmoid'

Iar în etapa finală am plecat de la modelul inițial și am adaugat după fiecare Conv2D un BatchNormalization și am mărit Dropout-ul la 50% alături de 15 epoci ce a condus la scorul 0.63022.

Layer	Hyperparameters
Conv2D	filters=32, kernel_size=(3, 3), activation='relu', padding='same', input_shape=(150, 150, 1)
BatchNormalization	
MaxPooling2D	pool_size=(2, 2)
Conv2D	filters=64, kernel_size=(3, 3), activation='relu', padding='same'
BatchNormalization	

Layer	Hyperparameters
MaxPooling2D	pool_size=(2, 2)
Conv2D	filters=128, kernel_size=(3, 3), activation='relu', padding='same'
BatchNormalization	
MaxPooling2D	pool_size=(2, 2)
Conv2D	filters=256, kernel_size=(3, 3), activation='relu', padding='same'
BatchNormalization	
MaxPooling2D	pool_size=(2, 2)
Flatten	
Dense	units=128, activation='relu'
BatchNormalization	
Dropout	rate=0.5
Dense	units=1, activation='sigmoid'

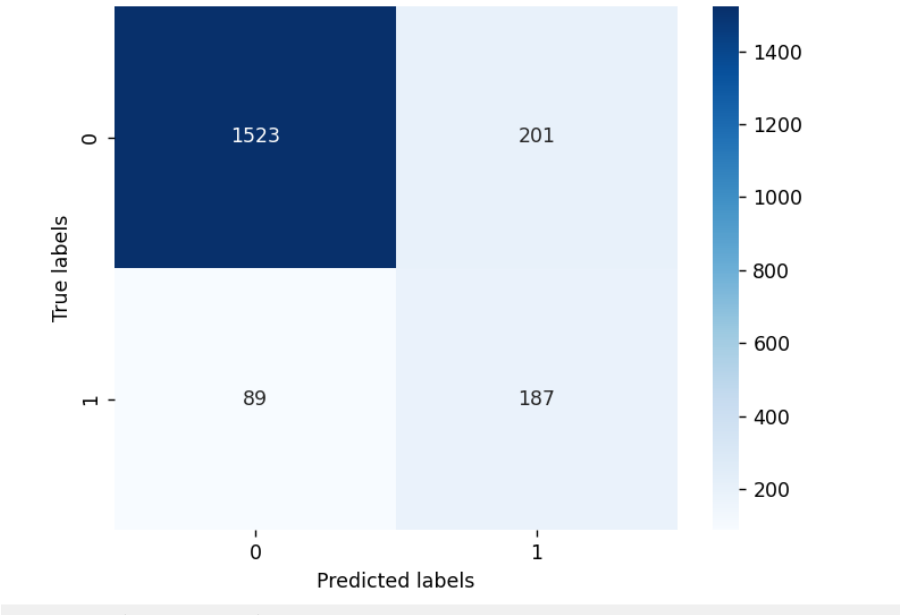

Fig.5 Matricea de confuzie pentru CNN final pe validation_data

Mihailescu Teodor
Grupa 242

	precision	recall	f1-score	support
0	0.94	0.88	0.91	1724
1	0.48	0.68	0.56	276
accuracy			0.85	2000
macro avg	0.71	0.78	0.74	2000
weighted avg	0.88	0.85	0.86	2000

Fig.6 Raportul pentru CNN final ce cuprinde accuracy rates, precision and recall for each class

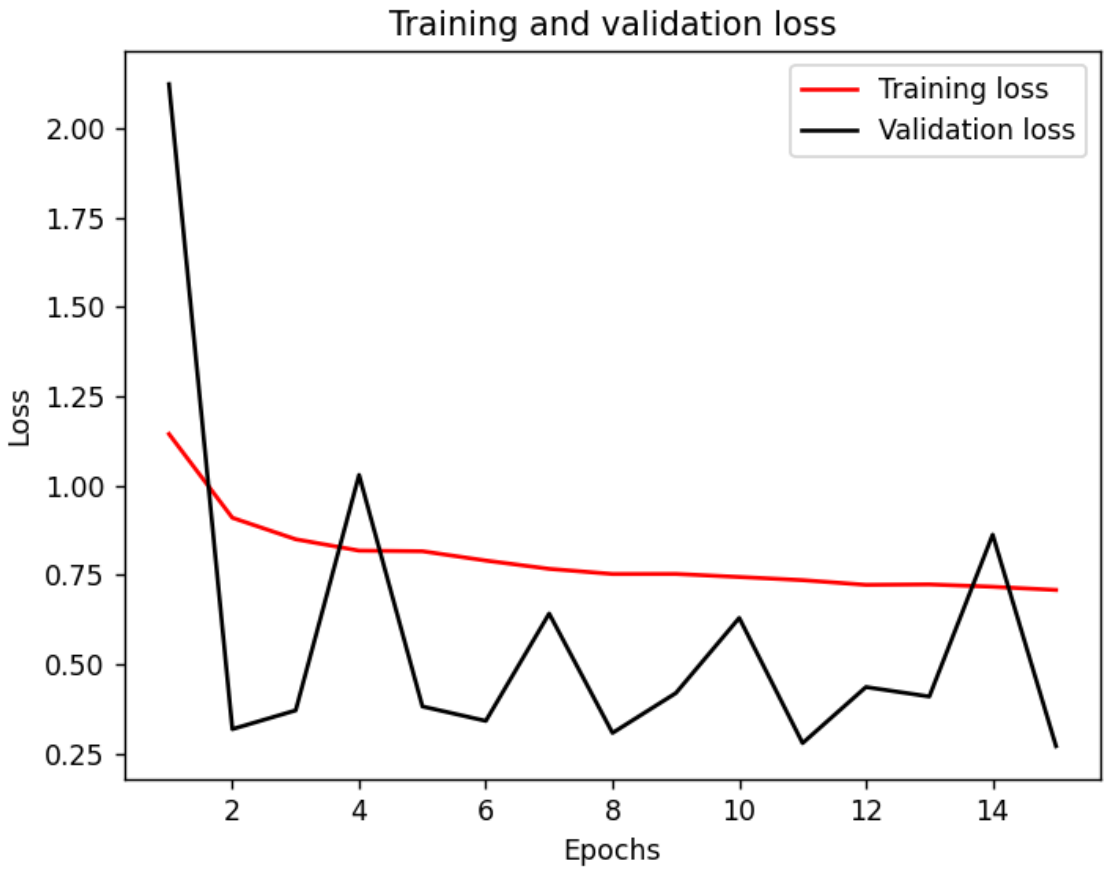

Fig.7 Training and Validation loss în cele 15 epoci