Ion Melania-Victorița
Grupa 241

# Brain Anomaly Detection

## *Descrierea proiectului*

Fiind dat un set de date care conține scanări CT ale creierului, proiectul constă în antrenarea unui clasificator pentru detectarea anomaliilor cerebrale. Mai exact, clasificarea acestor imagini de dimenisune 224x224 în 2 clase distincte: 0 (normal) si 1 (anomalie). Pentru clasficarea acestor imagini, primim un set de date ce constă în:

° 15,000 imagini pentru antrenare, împreună cu label-urile corespunzătoare
° 2,000 imagini pentru validare, însoțite de labu-rile corespunzătoare
° 5,149 imagini de test pentru competiție

## *Abordări*

Pentru a rezolva acest proiect, am abordat urmatorii algoritmi ce urmeaza a fi explicații în continuare: ° CNN
° Gaussian Naive Bayes

### 1. *CNN*

CNN (Convolutional Neural Network) este o categorie de rețele neuronale utilizată mai ales în prelucrarea imaginilor. Aceasta se compune dintr-o serie de straturi care procesează treptat informația, extrăgând caracteristici din ce în ce mai abstracte și ajungând astfel la o înțelegere profundă a imaginii. Această metodă permite rețelei să identifice și să clasifice cu acuratețe caracteristici complexe. Pentru crearea modelului, m-am folosit de Keras si Tensorflow.

*Procesare datelor si augmentarea acestora*

Pentru a îmbunătăți performanța modelului și pentru a evita probleme precum underfitting sau overfitting, am ales sa fac anumite procesări asupra setului de date.

Primul lucru pe care l-am facut a fost normalizarea datelor pentru a ajuta uniformizarea scalării acestora și reducerea impactului variațiilor inutile ale intensității pixelilor. Astfel, am împărțit valorile tuturor pixelilor din imagine cu 255, asigurând faptul că vom avea valori cuprinse între 0 și 1. În continuare, pentru a ajuta modelul sa învețe să recunoască și să se adapteze la o gamă mai largă de exemple pentru a crește capacitatea de generalizare, am decis sa efectuez urmatoarele augmentări:

- am utilizat layers.RandomFlip("horizontal") ce execută un flip orizontal aleatoriu asupra imaginilor în timpul fiecărei epoci de antrenare pentru a le varia poziția

- layers.RandomRotation (0.1) este o altă operație de augmentare a datelor pe care am utilizat-o ce aplică o rotație aleatoare cu un unghi maxim dat, iar valoarea pe care am ales-o este 0,1

Întrucât după prima încercare de antrenare nu am obținut rezultate foarte bune și pentru a diversifica și mai mult, am adăugat și operația zoom asupra imaginilor cu o valoare de 0.1 ce

reprezintă factorul maxim de zoom. Am ales această valoare în algoritmul final deoarece un număr mai mare (ex: 0,4) a avut un efect negativ asupra performanței modelului, conducând probabil la pierderea unor detalii importante din imagine.

## *Construirea modelului*

### *a) Layere utilizate*

Am utilizat clasa Sequential din biblioteca Keras pentru definirea modelului, ce permite adăugarea secvențială a straturilor. Primele două straturi adăugate au fost cele de normalizare și de augmentarea a datelor, definite anterior. Voi prezenta în continuare ce layere am folosit, în ordinea adăugată din varianta finală:

° Conv2D – primul strat este un strat convoluțional pentru detectarea caracteristicilor specifice ale imaginii ce utilizează 16 filtre cu dimensiunea kernel-ului de 3x3, cu funcția de activare ReLU și regularizarea L2 (cu valoarea 0.0001)

° BatchNormalization – acest strat normalizează valorile de ieșire ale stratului de convoluție prin scăderea mediei și împărțirea deviației standard

° MaxPooling2D - este un strat de max-pooling bidimensional ce are rolul de a reduce dimensiunile tensorului de intrare prin împărțirea în ferestre de 2x2 pixeli și reținerea valorii maxime din fiecare fereastră

° Flatten – layer care transformă tensorii de intrare multidimensionali într-un tensor unidimensional pentru a fi procesați de următorul strat adăugat ce este unul dens

° Dense - reprezintă un strat dens (fully connected) din rețeaua neuronală (fiecare neuron din stratul curent primește input-ul de la toți neuronii din stratul anterior)

° Dropout – are rolul de a preveni overfitting-ul prin eliminarea aleatorie a unor neuroni în timpul antrenării

### *b) Hiperparametrii*

Pentru layerele Conv2D am utilizat ca numar de filtre 16, 32, 64, crescând astfel pentru fiecare nou strat convolutional adăugat, în total fiind 3 în varianta finală. Dimensiunea kernel-ului va rămâne de 3x3, cu funcția de activare ReLU, padding = “same” astfel încât dimensiunile input-ului să fie aceleași cu ale output-ului și regularizarea L2. Am încercat valoarea 0,001 pentru regularizare, însă am obținut un rezultat mai bun cu o valoare mai mică de 0,0001.

Pentru primul layer Dense folosesc funcția de activare ReLU, aceeasi regularizare L2 și un număr de 128 de neuroni (units), în timp ce pentru ultimul utilizez un număr de 2 unități ce reprezintă numărul de clase (și regularizarea L2).

Utilizez un layer Dropout și am păstrat drept rata de abandon la 0,2 în varianta finală, după ce am încercat cu valorile 0,3 si 0,4 (aveam o acuratete mai mică cu aceste valori, ceea ce sugerează faptul că aveam de a face cu underfitting).

Pentru **compilarea modelului** se folosește funcția de loss SparseCategoricalCrossentropy pentru a observa cât de bine modelează valorile așteptate. Funcția compară distribuția probabilității prezise de model cu distribuția probabilității reale a etichetelor. Am ales-o drept functie de loss deoarece este frecvent folosită în probleme de clasificare atunci când etichetele

țintă sunt reprezentate printr-un set de întregi.  Drept optimizator, am uitlizat Adam (ce poate fi considerat o extindere a optimizatorului SGD, însă este mai rapid) și am încercat cu o rată de învățare de 0,0001, însă am păstrat rata de 0,001 în varianta cea mai bună. Metrica utilizată este acuratețea.

Am încercat antrenarea modelului timp de 5, 10, 15, 20 si 22 de epoci, însă varianta ce a obținut cel mai bun scor pe kaggle a fost antrenată timp de 20 de epoci.

## *Variantele modelului*

Aceasta este prima variantă încercată ce a obținut acuratețea de 0,26801 pe kaggle în care am folosit doar primele 2 tehnici de augmentare a datelor menționate anterior, o retea neuronală mai mică și a fost antrenată timp de doar 10 epoci.

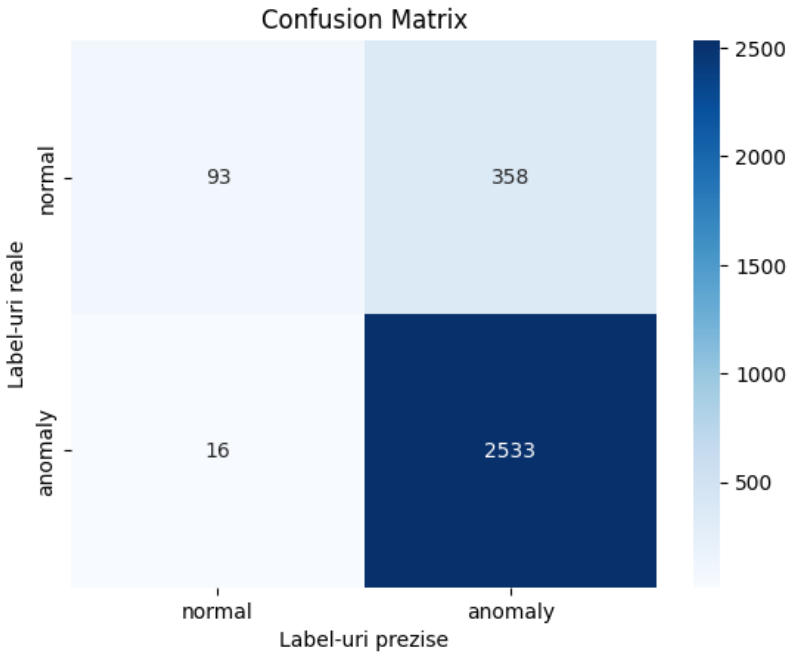
```
model = Sequential([
  data_augmentation,
  layers.Rescaling(1./255),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),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

Următoarea variantă este cea finală care a obținut un scor pe kaggle de 0,53212 în care a fost adăugată și operația de zoom pentru a crește capacitatea de generalizare. De asemenea, am mărit rețeaua neuronală prin adaugarea de noi layere precum cele convoluționale sau câte un layer BatchNormalization după fiecare dintre acestea pentru a normaliza valorile de output ale stratului anterior și a îmbunătăți acuratețea. În plus, pentru a preveni overfitting-ul am utilizat regularizarea L2 și am adăugat un layer Dropout astfel încât la fiecare pas al antrenării, o probabilitate fixă este aleasă pentru fiecare neuron din stratul anterior și în funcție de aceasta, neuronul este păstrat sau nu. Astfel, neuronii nu pot depinde de alți neuroni și vor trebui să învețe caracteristici care sunt utile independent de ceilalți.

```python
l2_lambda = 0.0001

#definirea modelului
model = Sequential([
layer_normalizare,
    augmentarea_datelor,
    layers.Conv2D(16, 3, padding='same', activation='relu', kernel_regularizer=tf.keras.regularizers.l2(l2_lambda)),
    layers.BatchNormalization(),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu', kernel_regularizer=tf.keras.regularizers.l2(l2_lambda)),
    layers.BatchNormalization(),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu', kernel_regularizer=tf.keras.regularizers.l2(l2_lambda)),
    layers.BatchNormalization(),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(128, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(l2_lambda)),
    layers.Dropout(0.2),
    layers.BatchNormalization(),
    layers.Dense(numar_clase, kernel_regularizer=tf.keras.regularizers.l2(l2_lambda))
])
```

*Matricea de confuzie:*



Classification Report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| normal | 0.85 | 0.21 | 0.33 | 451 |
| anomaly | 0.88 | 0.99 | 0.93 | 2549 |
| accuracy | 0.88 | 3000 | | |
| macro | avg | 0.86 | 0.6 | 0.63 |
| weighted | avg | 0.87 | 0.88 | 0.84 |

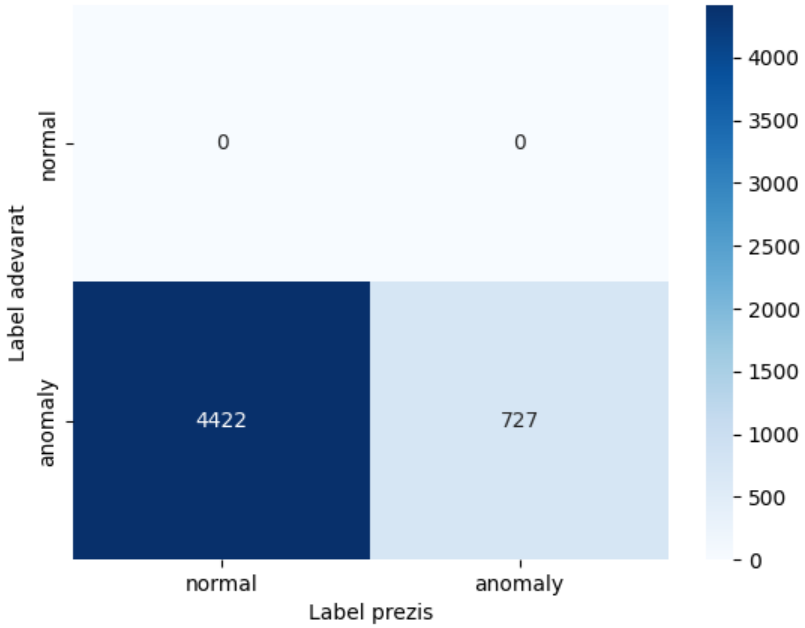
## 2. *Gaussian Naive Bayes*

Gaussian Naïve Bayes a fost prima abordare în rezolvarea problemei. Acesta este un algoritm de clasificare probabilistică ce se bazează pe teorema lui Bayes și presupune că toate caracteristicile datelor de intrare au aceeași distribuție normală și sunt complet independente.

Am împărțit datele în subdirectoarele "anomaly" și "normal" pentru a încărca imaginile din acestea în liste separate de imagini și etichete. Apoi, împart datele în setul de antrenare (80%) și cel de validare (%20) cu ajutorul funcției train_test_split() din din biblioteca Scikit-learn. Mă folosesc de parametrul “random_state” (fixez o valoare de referință) pentru a avea aceeași împărțire a datelor pentru diverse rulări ale codului.

În continuare, creez o instanță a modelului GNB și îl antrenez folosind setul de date de antrenare și etichetele lor corespunzătoare, astfel acesta va învață distribuțiile probabilităților clasei și caracteristicile, urmând a fi testat pe datele de test de pe kaggle.

În teorie, modelul ar trebui să fie simplu și rapid de antrenat, însă în ceea ce privește problema noastră, există mai multe limitări importante ale acestui algoritm. Una dintre acestea o reprezintă presupunerea caracteristicile de intrare a fi independente, ceea ce poate fi o problemă în cazul CT-urilor (există o multitudine de caracteristici care sunt puternic corelate între ele, cum ar fi densitatea și textura). Din aceste motive, am optat pentru retele neuronale convolutionale în rezolvarea temei și am încercat diverse variante pentru acesta, ceea ce a condus la obținerea unui rezultat mai bun.


normal
anomaly
Label adevarat
0
0
4422
727
normal
anomaly
Label prezis
4000
3500
3000
2500
2000
1500
1000
500
0