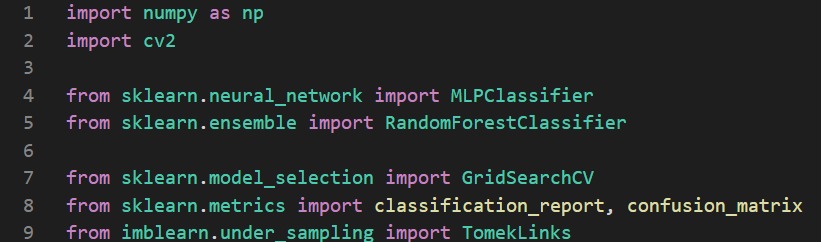
**DOCUMENTATIE**

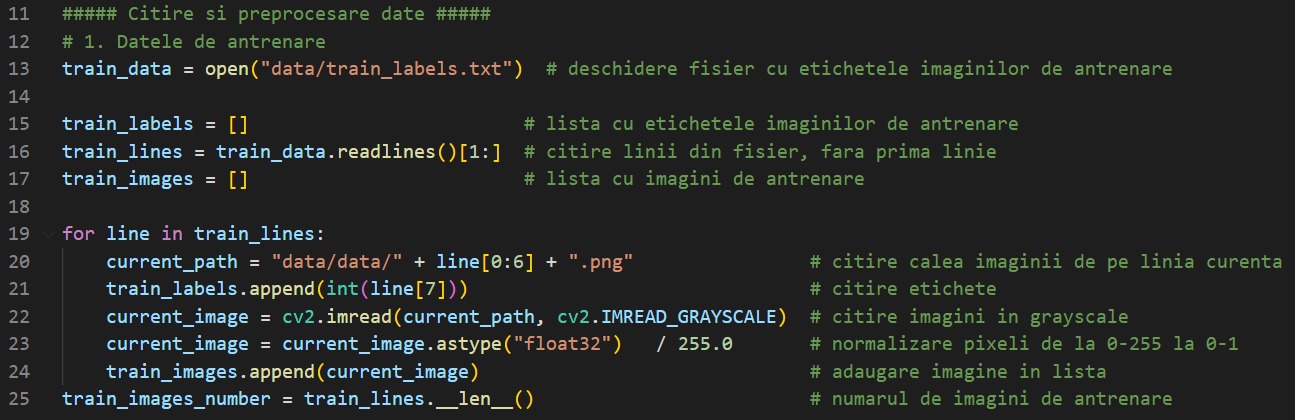
Brain Anomaly Detection – Proiect Învățare Automata

În proiect se vor clasifica scanări CT (Tomografie Computerizată) cerebrale în două clase diferite: imagini neconținând niciun fel de anomalie (clasa 0) sau cu diverse anomalii cerebrale (clasa 1). Putem rezolva problema printr-o învățare supervizată prin diverse metode bine-cunoscute.

În modelul de învățare, introducem un set de date pentru antrenare care sunt deja etichetate pentru a învăța modelul să clasifice predicții pentru seturi de date neetichetate. Problema va fi abordată în mai multe moduri pentru a găsi modelul care ne oferă cea mai bună precizie.

1. **Vom importa clasele utile folosite pentru implementarea soluției:**



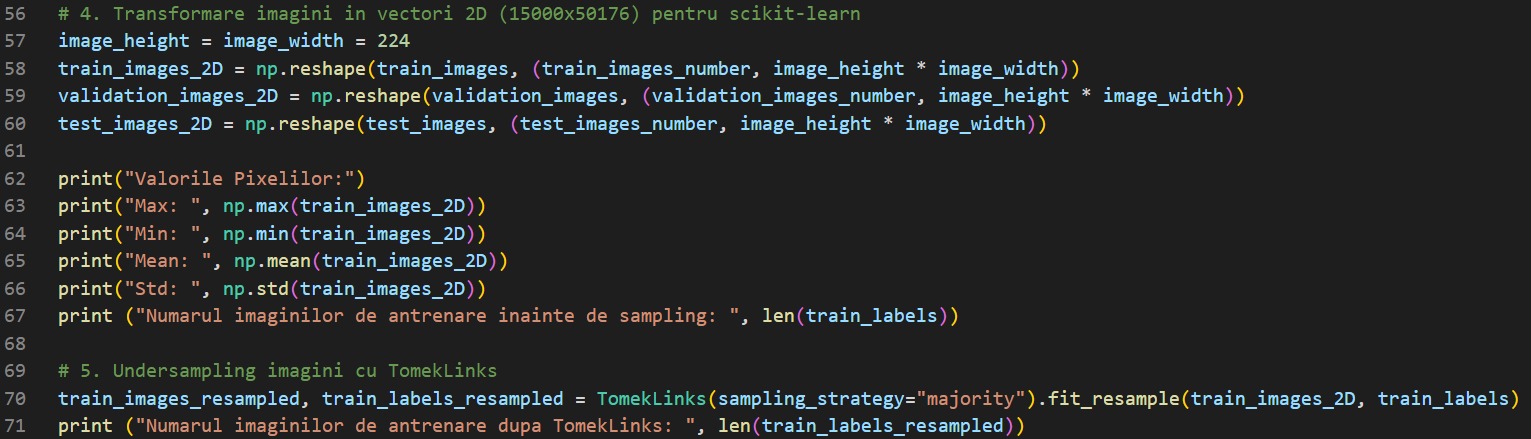
1. **Numpy** – bibliotecă utilă pentru funcții matematice.
2. **OpenCV (cv2)** – bibliotecă folosita pentru citirea imaginilor din fișiere.
3. **Sklearn-MLPClassifier** – bibliotecă care ne oferă modelul rețelei neuronale multi-strat Perceptron, utilizată pentru problemele de clasificare.
4. **SKlearn-RandomForectClassifier** – bibliotecă care ne oferă algoritmul RFC, funcționând prin arbori de decizie.
5. **Sklearn-GridSearchCV** – bibliotecă pentru folosirea Cross Validation, de asemenea ne găsește cei mai buni hiperparametrii dintr-o lista.
6. **Sklearn-Classification\_Report** – bibliotecă pentru obținerea raportului de clasificare, ne ajută sa găsim modelul cel mai bun.
7. **Sklearn-Confusion\_Matrix** – bibliotecă pentru obținerea matricei de confuzie.
8. **ImbalancedLearn-TomekLinks** – bibliotecă folosită pentru a elimina conexiunile Tomek.
9. **Citim și prelucrăm seturile de date pe care vom lucra:**

Aceeași metodă a fost folosită și pentru datele de validare, și pentru cele de testare. Exceptând faptul că setul de testare nu deține etichetele, deci nu le vom salva.

Începem prin deschiderea fișierului care conține datele pentru antrenare, unde vom obține numele imaginii în combinație cu eticheta respectivă.

Inițiăm doi vectori, unul pentru imagini și unul pentru etichete. Vom reține toate liniile din fișierul de date excluzând primul (care definește numele coloanelor) în variabila „train\_lines”. Apoi pentru fiecare linie, luăm calea imaginii, introducem eticheta imaginii în vectorul cu etichete, citim imaginea bazată pe calea ei în alb-negru și normalizăm pixelii să aibă o valoare în intervalul [0, 1], în loc de [0, 255].

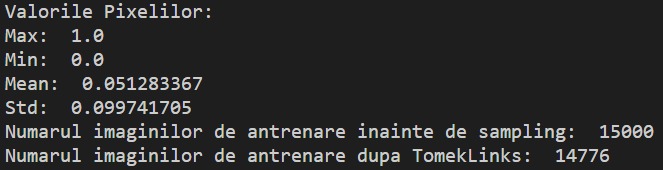
După toate aceste operații, introducem imaginea în vectorul cu imagini. Salvăm numărul imaginilor într-o variabilă, vom avea nevoie mai târziu de el.

1. **Mai multă prelucrare asupra datelor**

În primul rând, pentru a putea folosi aceste date cu bibliotecile oferite de Sklearn, vom avea nevoie să le transformăm în vectori bidimensionali.

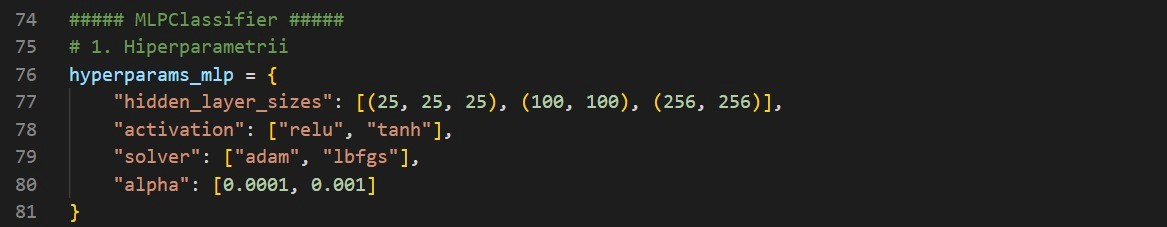
Știm că dimensiunile unei imagini sunt 224x224 pixeli, deci vom face două variabile egale unde vom defini lățimea și lungimea imaginii.

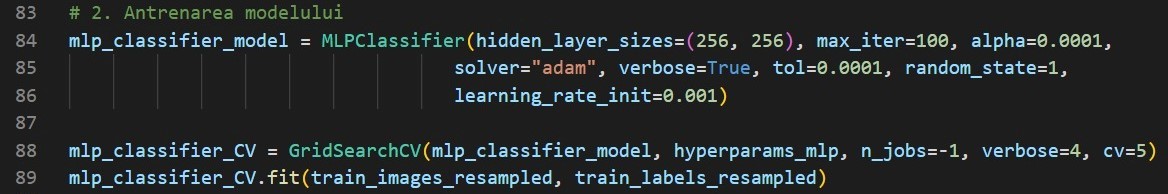
După ce avem aceste două variabile, remodelăm toți cei 3 vectori conținând imaginile pentru fiecare set de date din vectori tridimensionali de tip <nr\_img>x224x224 în <nr\_img>x50176, bineînțeles unde nr\_img este numărul imaginilor din fiecare set de date.

Afișăm pe ecran date în privire la imaginile de testare, precum valoarea maximă și minimă a pixelilor, media valorilor și deviația standard a acestora.

După informațiile pixelilor, afișăm numărul imaginilor de antrenare inițiale și încă o dată după ce folosim o metodă de eliminare a conexiunilor Tomek. O conexiune Tomek constă în două noduri care sunt vecinul cel mai apropiat unul altuia, însă aparțin unor clase diferite.

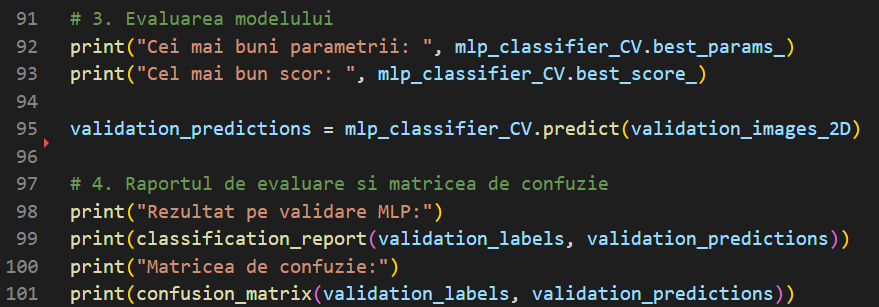
Prin urmare, după afișarea numărului de imagini inițiale, vom elimina aceste conexiuni prin eliminarea imaginilor din clasa majoritară, pentru a obține o clasificare mai marginală.

1. **Stabilim hiperparametrii folosiți pentru antrenarea modelului Perceptron**

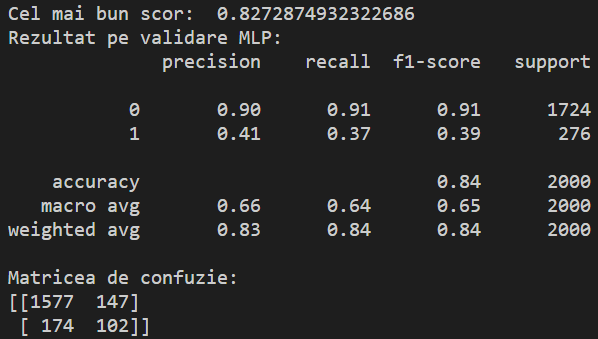
****

GridSearchCV ne-a ajutat să găsim hiperparametrii care ne oferă cele mai bune rezultate pentru modelul MLP (MultiLayerPerceptron), care ulterior sunt folosiți de fiecare dată în antrenarea modelului. Ce ne mai oferă și funcția GridSearchCV este „Cross Validation” (CV în numele funcției), care constă în folosirea unor porți diferite de date din cele de antrenare pentru a testa și antrena modelul.

În cazul nostru, se va face de cinci ori validarea.

1. **Testarea și evaluarea modelului nostru MLP**

După antrenare, avem nevoie să evaluăm modelul nostru pentru obținerea celor mai bune rezultate. Vom afișa parametrii cei mai optimi găsiți de GridSearchCV și cel mai bun scor oferit de aceștia.

Cu acest model, vom face predicții pentru setul de date de validare, care ne oferă o acuratețe mai bună asupra performanței modelului.

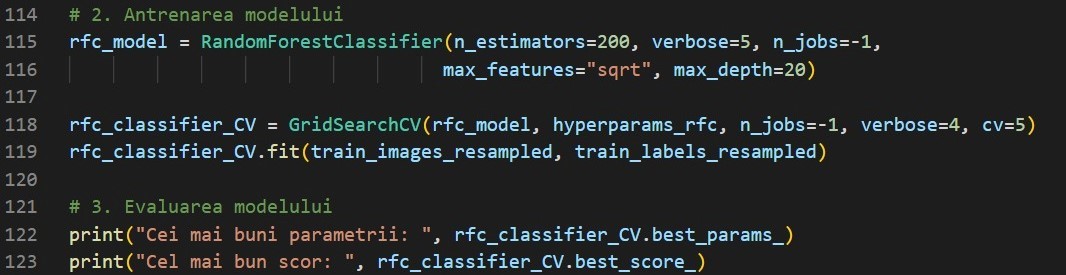
Cel mai bun scor obținut de GridSearchCV poate fi observat în imaginea de mai sus. O precizie de 82%, dar cam inutilă dacă modelul nostru conține 82% de imagini într-o singură clasă.

Raportul de clasificare ne ajută să privim mai bine performanța modelului prin mai multe categorii de scor: precision, recall, f1-score și support. Ideal ar fi ca primele trei categorii să fie cât mai ridicate. F1-score este o medie a preciziei și recall-ului și este cam cea mai importantă categorie.

O altă metodă bună de clasificare este matricea de confuzie. Din câte se poate observa, avem 1577 imagini clasificate corect în clasa 0 și 102 în clasa 1. Par a fi multe, însă „support” ne arată numărul real de imagini în fiecare categorie.

Avem 1577 din 1724 în clasa 0 și 102 din 276 în clasa 1. Modelul nostru clasifică 174 imagini ca fiind negative în loc de pozitive și 147 ca fiind pozitive în loc de negative.

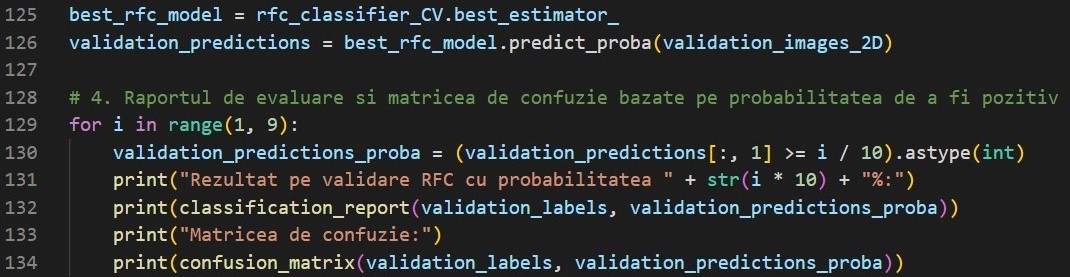
1. **Găisrea hiperparametrilor optimi și antrenarea modelului Random Forest**

****

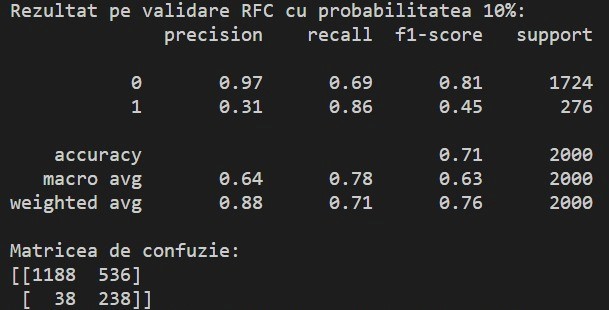
Asemenea modelului anterior MLP, vom face aceleași operațiuni și cu modelul Random Forest. De această dată, evident, hiperparametrii vor fi diferiți.

Numărul de iterații pentru Cross Validation rămâne același. Un avantaj bun al folosirii funcției GridSearchCV este că poate utiliza toate nucleele procesorului pentru a crește viteza procesului.

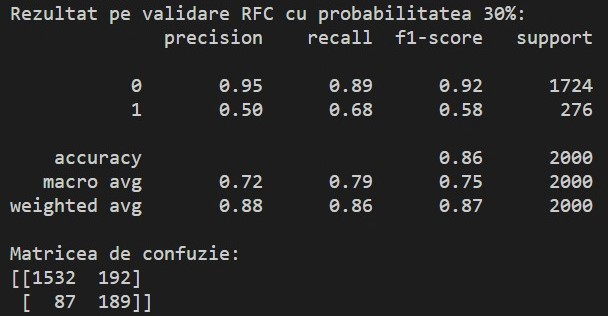
Putem vedea parametrii și scorul cel mai bun găsite de funcția GridSearchCV în imaginea de mai sus.

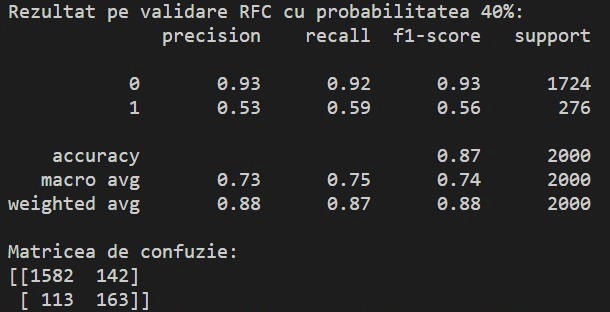
1. **Testarea și evaluarea modelului nostru Random Forest**

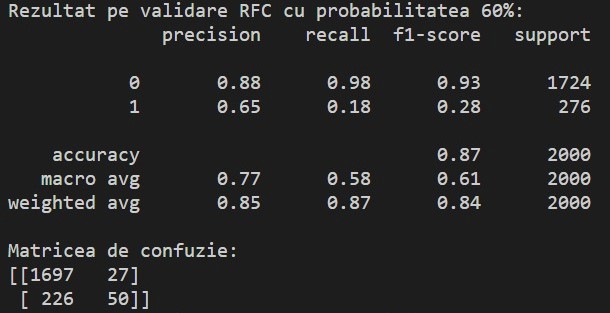
Vom folosi o abordare diferită pentru evaluarea modelului Random Forest. În loc să folosim clasificări hard 0-1, vom categoriza imaginile bazate pe probabilitatea ca o imagine să fie pozitivă. În cazul unui set de date greu și neechilibrat, ne oferă un control mai bun asupra etichetării.

Vom testa probabilități de la 10% la 80% ca o imagine să fie clasificată pozitiv. Nu voi arăta toate cele opt teste.

Probabilitatea de 10% ca o imagine să fie tratată ca fiind pozitivă ne oferă, bineînțeles, un număr extrem de mare de fals pozitive. La 20%, situația începe să fie mai acceptabilă, dar tot este exagerată.



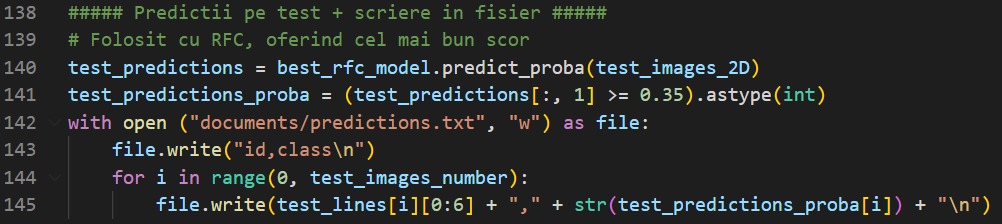
Probabilitatea de 30% oferă un scor destul de bun pentru clasa pozitivă. Nu este ideal, dar nici foarte rău, precum la 10% sau cel mai mult la 80%. Rezultatul final folosit în competiție este, de fapt, o probabilitate de 35%.



Probabilitatea de 60% deja clasifică foarte multe imagini drept fals negative, mai departe de atât doar reduce semnificativ numărul de imagini clasificate în clasa pozitivă.

Aceasta este metoda folosită pentru clasificarea imaginilor folosind modelul Random Forest. Este modelul care oferă cea mai bună precizie dintre modelele folosite.\

1. **Fișierul de predicții folosit in cadrul competiției, cu setul de date de testare**

****

Pentru clasificarea predicțiilor cu setul de date de testare, vom folosi o probabilitate de 35% pentru a clasifica imaginile pozitive.

Scrierea fișierului este realizată cu ajutorul funcțiilor obișnuite din limbajul Python. Fișierul este în format .txt, iar ulterior va fi convertit manual în format .csv.