Gonțescu Maria Ruxandra
Grupa 241

# Brain anomaly detection

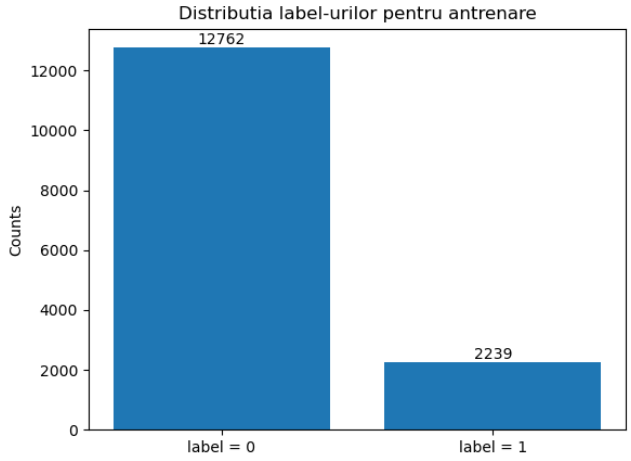
## I. Descrierea proiectului

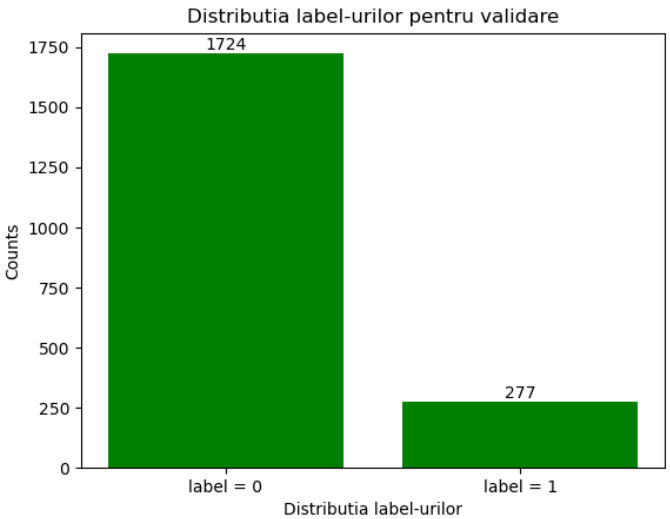
Proiectul presupune clasificarea unor imagini de dimensiune 224*224*3 ce reprezintă scanări CT ale creierului. Aceste tomografii trebuie clasificate binar în 0 (sănătos) sau 1(bolnav).

## II. Setul de date

Pentru a putea clasifica aceste imagini, se primesc ca input următoarele date:

- 15000 de imagini pentru antrenare, însoțite de label-urile corespunzătoare. Imaginile sunt în format .PNG, iar label-urile sunt citite din fișierul “train_labels.txt”.
- 2000 de imagini pentru validare, precum și label-urile acestora. Label-urile pentru validare sunt citite din fișierul “validation_labels.txt”.
- 5149 imagini de test.



Gonțescu Maria Ruxandra
Grupa 241

# III. Abordări

În rezolvarea acestei probleme, am abordat 2 strategii. Un prim model încercat a fost cel bazat pe Naive Bayes. A doua abordare a fost folosirea rețelelor neuronale convolutionale.

## A. Naive Bayes

Inițial am optat pentru Naive Bayes. Naive Bayes este un un algoritm probabilist bazat pe teorema lui Bayes. Clasificatorul calculează probabilitatea ca o ipoteza sa fie adevărată pe baza cunoștințelor anterioare. Acesta presupune că toate caracteristicile sunt independente unele de celelalte, de aceea este ”naiv”.

In acest prim model nu am realizat transformări majore asupra datelor de intrare. Pentru a citi datele am definit o functie “read_image”. Este utilizată metoda “Image.open()” din Python Imaging Library (PIL). Aceasta primește ca argument concatenarea dintre path-ul imaginii curente și ID-ul acesteia. Imaginile au un format de 224 * 224 * 3. Dimensiunile de 224 * 224 se referă la înălțimea și lățimea acestora, în timp ce 3 semnifica numărul de de canale de culoare. Ulterior, transformam fișierul într-un format greyscale ( 224 * 224 * 1) prin intermediul metodei “.convert(‘L’)”, deoarece am dorit o scădere a complexitatii (modelul devine mai rapid și mai ușor de antrenat). In final, imaginea este transformată în nparray cu ajutorul “img_to_array(image)” din modulul “keras.preprocessing.image”.

Citirea este urmată de normalizare. Am calculat media și deviația standard a fiecărei coloane ( fiecare pixel) utilizând funcțiile NumPy “np.mean” și “np.std”. Fiecare pixel este normalizat prin scăderea mediei și împărțirea la deviația standard. Rezultatul este un tablou NumPy cu aceleași dimensiuni, dar cu valorile normalizate pentru fiecare pixel. In plus, ne-am asigurat de faptul că datele au formatul(shape-ul) valid pentru următoarele transformări.

În procesul de cuantificare a valorilor sunt utilizate funcțiile “get_intevals(num_bins)” și “values_to_bins(x,bins)”. “Get_intervals” generează un set de “num_bins” intervale de valori de la 0 la 256. În urma mai multor încercări am ajuns la valoarea optima num_bins = 5. “Values_to_bins” asociază fiecare valoare a pixelilor cu intervalul corespunzător în care se încadrează.

Înainte de a fi pasate clasificatorului MultinomialNB, datele au fost scalate astfel încât sa se încadreze în intervalul [0, 1].

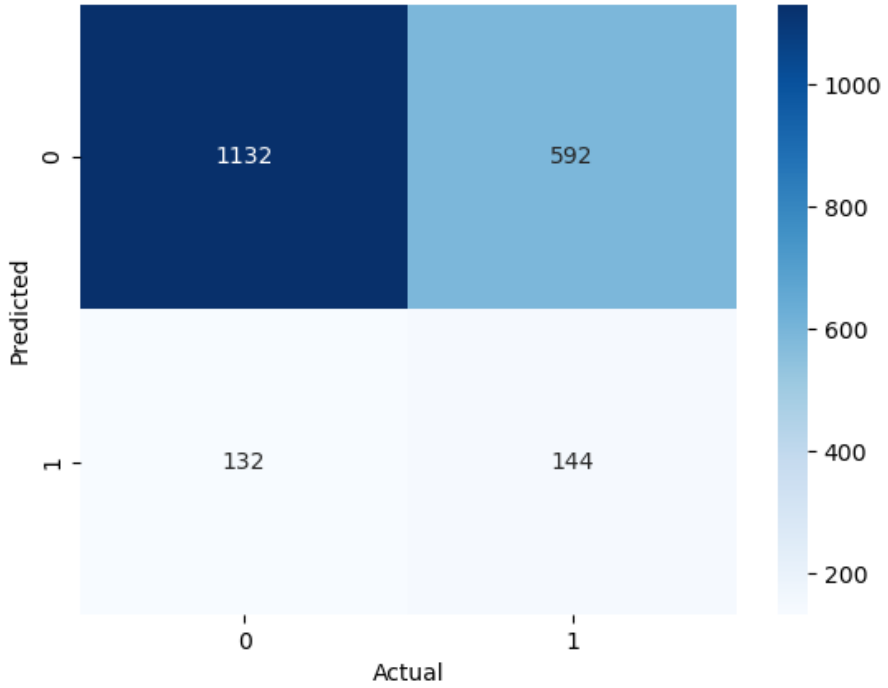
Acest model a obținut pe validare un procent de 63, 8%, iar scorul f1 a fost. 0.334.

F1 score 0.334
Precision 0.195
Recall 0.521
Matricea de confuzie:

(1132 592)

( 132 144)

Gonțescu Maria Ruxandra
Grupa 241




Astfel,

- 1132 reprezintă numărul de exemple din clasa negativă care au fost clasificate corect (true negatives)
- 592 reprezintă numărul de exemple din clasa pozitivă care au fost clasificate incorect (false negatives)
- 132 reprezintă numărul de exemple din clasa negativă care au fost clasificate incorect (false positives)
- 144 reprezintă numărul de exemple din clasa pozitivă care au fost clasificate corect (true positives)

În cazul lui num_bins, am încercat mai multe posibilități, iar rezultatele obținute au fost:

- num_bins = 7 => 64,35% procent pe validare, f1 = 0.312,
  Precision 0.202
  Recall 0.547
  [[1128 596]. Matricea de confuzie pentru num_bins = 7
  [ 125 151]]

- num_bins = 8 => 68,3 % procent pe validare, f1 = 0.298
  Precision = 0.2039
  Recall = 0.5579
  [[1123 601]

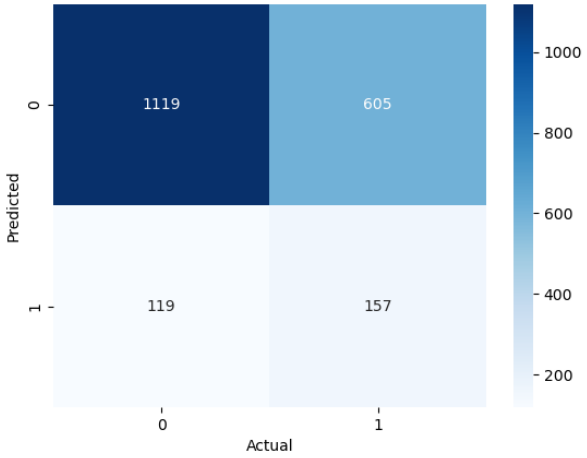
[ 122  154]]

- num_bins = 9 => 74,5 % procent pe validare,  f1 =  0.30

Precision = 0.20657894736842106
Recall =  0.5688405797101449
[[1121  603]
[ 119  157]]

- num_bins = 10 => 76,9 % procent pe validare,  f1 =  0.3025

Precision = 0.206
Recall =  0.568
[[1119  605]
[ 119  157]]




## B. Rețele neuronale convoluționale(CNN)

Rețelele neuronale convoluționale se aseamănă cu rețelele neuronale obișnuite, sunt alcătuite din neuroni care au weights-uri și bias-uri învățabile. Acestea cuprind straturi de convoluție peste care se aplica funcții de activare. În implementarea acestui model am utilizat PyTorch.

La fel ca la primul model, am citit, normalizat si transformat datele în tensori. În plus, la aceasta soluție am realizat și data augmentation.  Am ales transformarea datelor de intrare prin intermediul librăriei torchvision.transforms, am rotit imaginile la 15°, aleator s-au inversat orizontal, am ajustat luminozitatea, nuanța, saturația și contrastul. Dupa aceea, datele au fost convertite la tensori și normalizate.

Am pornit cu următoare arhitectura:

I.

```
class CTClassifier(nn.Module):

    def __init__(self):
        super(CTClassifier, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1)
        self.pool1 = nn.MaxPool2d(2, stride=2)
        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
        self.pool2 = nn.MaxPool2d(2, stride=2)
        self.conv3 = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1)
        self.pool3 = nn.MaxPool2d(2, stride=2)
        self.conv4 = nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1)
```

Gonțescu Maria Ruxandra
Grupa 241

```
        self.pool4 = nn.MaxPool2d(2, stride=2)
        self.conv5 = nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1)
        self.pool5 = nn.MaxPool2d(2, stride=2)
        self.fc1 = nn.Linear(7 * 7 * 512, 4096)
        self.fc2 = nn.Linear(4096, 1000)
        self.fc3 = nn.Linear(1000, 2)
        self.act = nn.ReLU()
```

```
def forward(self, x):
        x = x.view(-1, 224, 224, 3)    # different shape -  no problems for output
        x = torch.permute(x, (0, 3, 1, 2))   # change the order of dimensions of the
        x = self.conv1(x)
        x = self.act(x)
        x = self.pool1(x)
        x = self.conv2(x)
        x = self.act(x)
        x = self.pool2(x)
        x = self.conv3(x)
        x = self.act(x)
        x = self.pool3(x)
        x = self.conv4(x)
        x = self.act(x)
        x = self.pool4(x)
        x = self.conv5(x)
        x = self.act(x)
        x = self.pool5(x)
        x = torch.flatten(x, start_dim=1)
        x = self.fc1(x)
        x = self.act(x)
        x = self.fc2(x)
        x = self.act(x)
        x = self.fc3(x)
       return x
```

- Conv2d
  - implementează operația de convoluție bidimensională pentru imagini prin care se extrag caracteristici din imagini
  - Este configurat cu un număr de parametrii
  - De exemplu:
    - self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1)
    - Primește o imagine cu 3 canale de culoare ca intrare si produce 64 de filtre convolutionale ca ieșire. Acest strat are un kernel (filtru) de dimensiune 3x3, ceea ce înseamnă că se va deplasa peste imaginea de intrare în fereastra de 3x3 pixeli. Pașii de deplasare sunt setați la 1 în ambele direcții. Padding-ul este setat la 1.

Gonțescu Maria Ruxandra
Grupa 241

- MaxPool2d
  - implementează operația de pooling de tip max pentru imagini
  - De exemplu:
    - self.pool1 = nn.MaxPool2d(2, stride=2)
    - Constructorul clasei primește 2 argumente: dimensiunea ferestrei de pooling și pașii de deplasare pe verticală și orizontală între ferestrele de pooling. În acest caz, dimensiunea ferestrei este de 2x2, ceea ce înseamnă că ferestrele de pooling sunt de dimensiune 2x2, iar pașii de deplasare sunt de 2 pixeli. Astfel, imaginea de intrare este redusă la jumătate ca dimensiune datorită operației de pooling, iar numărul de caracteristici (filtre) este păstrat la fel.

- Linear
  - implementează un strat liniar într-o rețea neuronală, care realizează o transformare liniară a caracteristicilor de intrare
  - De exemplu:
    - self.fc1 = nn.Linear(7 * 7 * 512, 4096)
    - Constructorul clasei primește două argumente: dimensiunea caracteristicilor de intrare și dimensiunea caracteristicilor de ieșire. În acest caz, dimensiunea caracteristicilor de intrare este de 7x7x512, ceea ce înseamnă că aceasta este forma matricii de intrare a stratului, iar dimensiunea caracteristicilor de ieșire este 4096.
- ReLU
  - "Rectified Linear Unit" este aplicată ca funcție de activare după stratul `nn.Linear`, adică ieșirea stratului linear este trecută prin funcția ReLU. Aceasta poate ajuta la îmbunătățirea performanței rețelei neuronale prin introducerea unei componente neliniare în model.

Ca functie obiectiv am folosit “nn.CrossEntropyLoss”, care generează o valoare de pierdere care este utilizată pentru antrenarea rețelei neuronale.

Am folosit optimizatorul “Adam”, ce construiește un optimizer care poate fi utilizat pentru a antrena modelul și pentru a minimiza valoarea funcției obiectiv prin ajustarea parametrilor acestuia. Acesta primește ca parametru learning rate-ul lr = 0.001 ( hiperparametru optim). Learning rate-ul crescut conducea la overfitting.

Modelul curent a fost antrenat pe 20 de epoci si a obținut scorul de 0.46947 în platforma. Precision = 0.6211, Recall = 0.41167, procent de validare 86.5%.

II.

Ulterior am introdus în cod un nou parametru în cadrul optimizatorului Adam, weight_decay = 1e-4 (pentru regularizarea L2, care ajută la prevenirea overfitting-ului prin reducerea magnitudinii parametrilor).

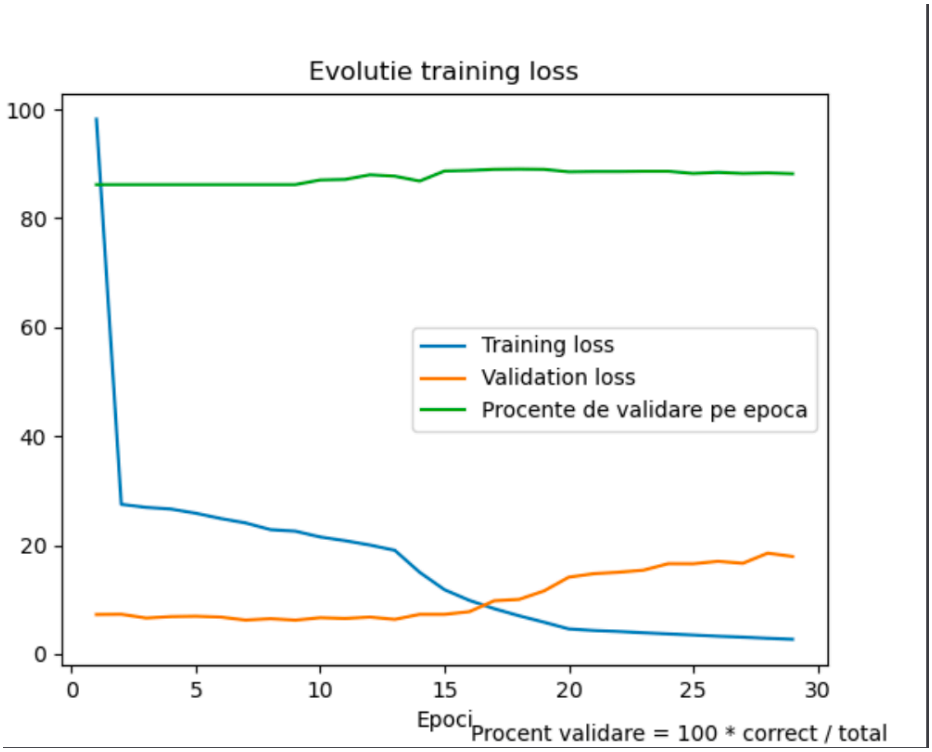
“ReduceLROnPlateau” este o strategie de programare a ratei de învățare atunci când loss-ul nu se îmbunătățește după un anumit număr de epoci. Parametrii optimi găsiți au fost:

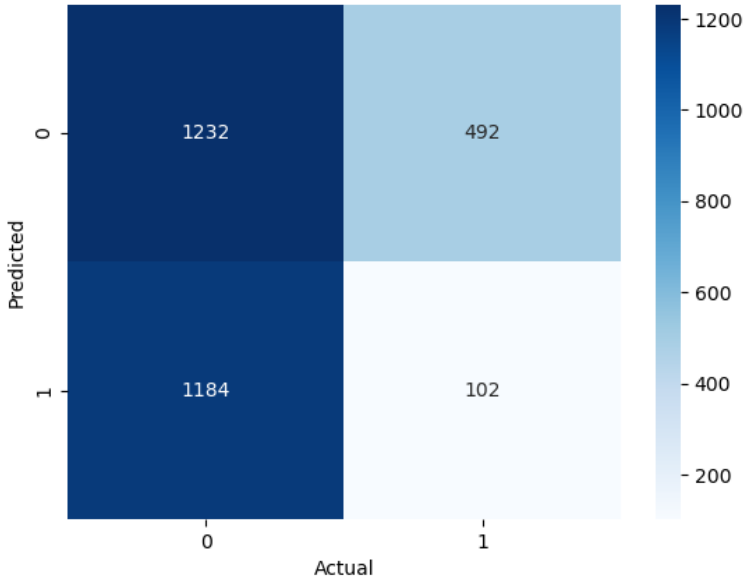
| optimizer | patience | min_lr | threshold | verbose | factor | Scor |
|---|---|---|---|---|---|---|

Gonțescu Maria Ruxandra
Grupa 241

| | | | | | | |
|---|---|---|---|---|---|---|
| Lr = 0.001, weight_decay = 1e-4 | 5 | 0.00001 | 0.01 | TRUE | 0.05 | 0,547 |
| Lr = 0.01, weight_decay = 1e-4 | 5 | 0.00001 | 0.01 | TRUE | 0.05 | 0,520 |

În urma acestor modificări am obținut scorul f1 0.547. Precision = 0.5519, Recall = 0.6238, procent de validare 88.14%.



Gonțescu Maria Ruxandra
Grupa 241

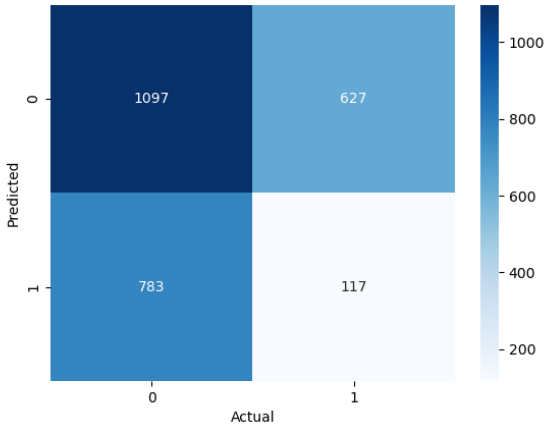
III.

Am încercat SMOTE(Synthetic Minority Over-sampling Technique) ca tehnica de oversampling pentru a echilibra distribuția claselor în setul de date. In urma generării de noi imagini ajunsesem la un total de 22000 de imagini. Se ajunsese la overfit. Această variantă a obținut un scor de 0.41 ( scădere a performantei). In concluzie, am abandonat aceasta varianta. Precision = 0.35299, Recall = 0.5106, procent validare = 86.9 %.

IV.

Pentru contrabalansarea celor 2 clase am introdus parametrul weight din cadrul funcției “CrossEntropyLoss”. Acesta atribuie o greutate mai mare clasei minoritare și o greutate mai mică claselor majoritare. Acest lucru ajută modelul să se concentreze mai mult asupra clasei minoritare și să evite supraestimarea claselor majoritare.

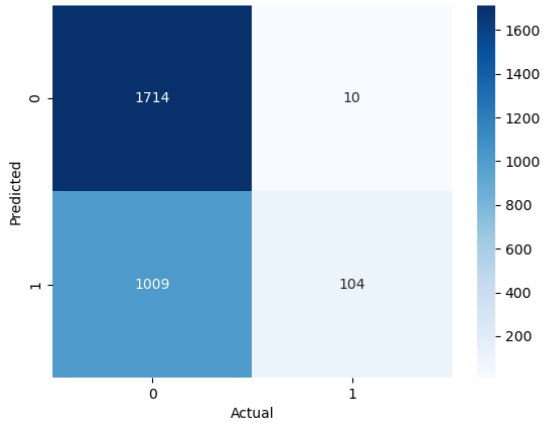
Rezultatul a fost de 0.6166. Precision = 0.7926, Recall = 0.541, procent validare = 89.2%.



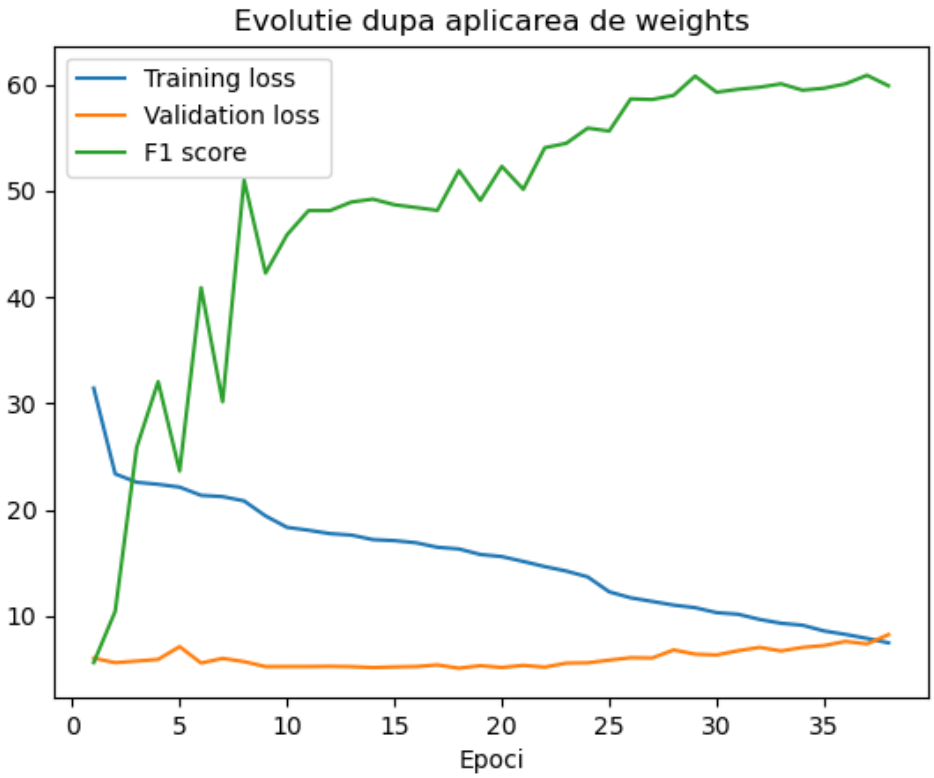

V.

Am dublat primul layer convoluțional din arhitectură. Scopul acestui layer este de a aplica un nou set de filtre rezultatului obținut în urma primului layer. Această strategie ajuta la învățarea caracteristicilor mai complexe ale datelor de antrenare.

Rezultatul a fost 0.6239. Precision = 0.601, Recall = 0.75, procent validare = 89.4%.



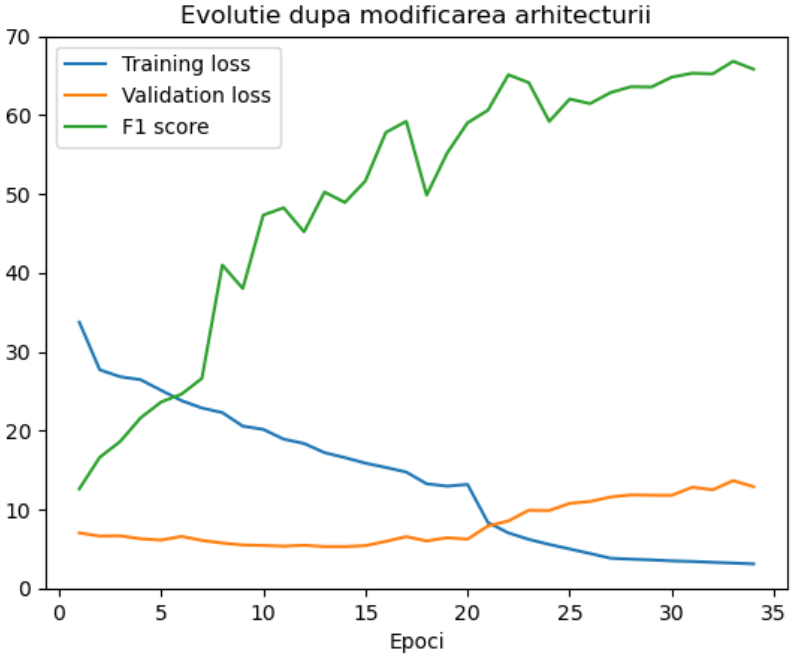
Gonțescu Maria Ruxandra
Grupa 241




VI.

Am observat o creștere a performantei, așa ca am ales sa dublez si al doilea layer convoluțional. Între cele 2 straturi convoluționale am adăugat funcție de activare. Aceasta permite introducerea neliniaritatilor în rețea si propagarea gradienților (backpropagation).

Rezultatul a fost 0.68852. Precision = 0.872, Recall = 0.593, procent validare = 89.9%.



Gonțescu Maria Ruxandra
Grupa 241

| | |
|---|---|
| 1687 | 37 |
| 1459 | 102 |

VII.

Varianta finala conține batch-normalization, utilizat pentru normalizarea datelor primite ca input. Acest lucru ajută la prevenirea problemelor precum gradientul care explodează sau dispare în timpul antrenamentului și accelerează procesul de antrenare al rețelei.

În plus, am adăugat si un drop-out ca tehnica de regularizare.

Rezultatul a fost de 0.72874. Precision = 0.85, Recall = 0.654, procent validare = 91.43%.

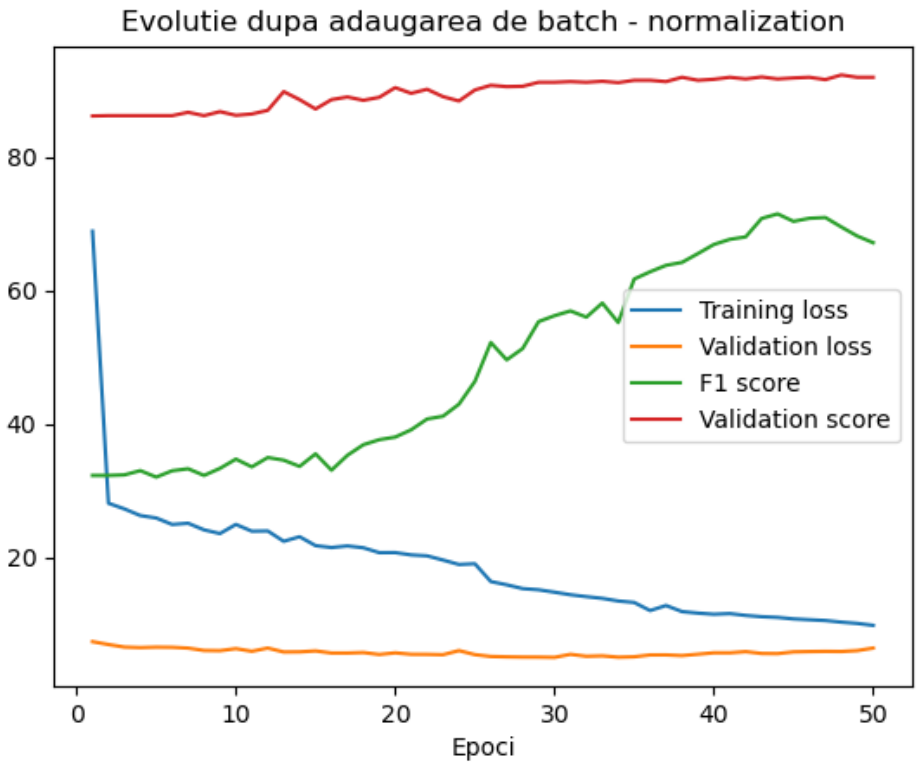
În final, arhitectura arata așa:

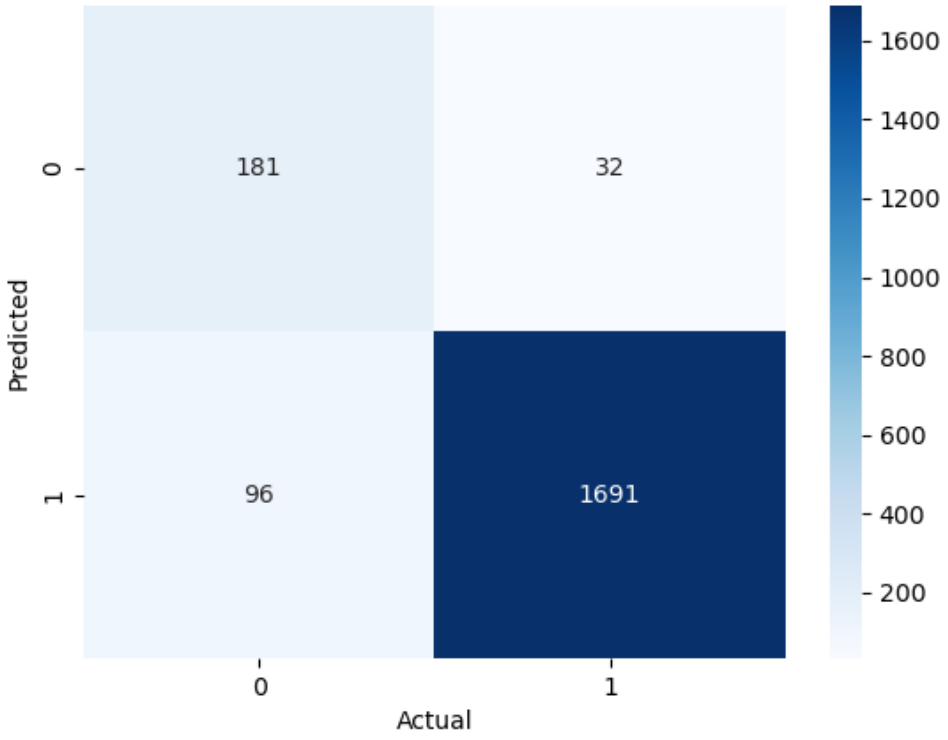
```
class CTClassifier(nn.Module):
    def __init__(self):
        super(CTClassifier, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1)
        self.conv12 = nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1)

        self.bn1 = nn.BatchNorm2d(num_features=64)
        self.pool1 = nn.MaxPool2d(2, stride=2)
        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
        self.conv22 = nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(num_features=128)
        self.pool2 = nn.MaxPool2d(2, stride=2)
        self.conv3 = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1)
        self.bn3 = nn.BatchNorm2d(256)
        self.pool3 = nn.MaxPool2d(2, stride=2)
        self.conv4 = nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1)
        self.bn4 = nn.BatchNorm2d(512)
        self.pool4 = nn.MaxPool2d(2, stride=2)
        self.conv5 = nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1)
        self.bn5 = nn.BatchNorm2d(512)
        self.pool5 = nn.MaxPool2d(2, stride=2)
        self.fc1 = nn.Linear(7 * 7 * 512, 4096)
        self.fc2 = nn.Linear(4096, 1000)
        self.dropout = nn.Dropout(0.3)
        self.fc3 = nn.Linear(1000, 2)
        self.act = nn.ReLU()

    def forward(self, x):
        x = x.view(-1, 224, 224, 3)    # different shape -  no problems for output
```

Gonțescu Maria Ruxandra
Grupa 241

```
        x = torch.permute(x, (0, 3, 1, 2))   # change the order of dimensions of the
tensor
        x = self.conv1(x)
        x = self.conv12(x)
        x = self.bn1(x)
        x = self.act(x)
        x = self.pool1(x)
        x = self.conv2(x)
        x = self.act(x)
        x = self.conv22(x)
        x = self.bn2(x)
        x = self.act(x)
        x = self.pool2(x)
        x = self.conv3(x)
        x = self.bn3(x)
        x = self.act(x)
        x = self.pool3(x)
        x = self.conv4(x)
        x = self.bn4(x)
        x = self.act(x)
        x = self.pool4(x)
        x = self.conv5(x)
        x = self.bn5(x)
        x = self.act(x)
        x = self.pool5(x)
        x = torch.flatten(x, start_dim=1)
        x = self.fc1(x)
        x = self.act(x)
        x = self.fc2(x)
        x = self.act(x)
        x = self.dropout(x)
        x = self.fc3(x)
        return x
```

Gonțescu Maria Ruxandra
Grupa 241




Bibliografie:

1. https://pytorch.org
2. https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html
3. https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

Gonțescu Maria Ruxandra
Grupa 241

4. https://pillow.readthedocs.io/en/stable/
5. https://www.tensorflow.org/api_docs/python/tf/keras
6. https://github.com/Ruxi12/An-2/blob/main/Semestrul%202/IA/ML/Proiect/rezultate.txt