Documentație proiect Machine Learning - Horceag Cristian-Andrei, 241

I. Descrierea Proiectului

În cadrul proiectului, scopul final a fost clasificarea unor imagini ce reprezintă scanări ale creierului uman în două clase (numerotate 0 și 1), pentru a identifica dacă pacientul prezintă o anomalie sau nu.

II. Setul de date

- ***imagini_train*** - reprezintă imaginile din setul de antrenare, primele 15000 de imagini din directorul data
- ***train_labels*** – reprezintă clasele fiecărei imagini din setul de antrenare(0 - dacă pacientul nu prezintă anomalie, 1 în caz contrar)
- ***imagini_validation*** - reprezintă imaginile din setul de validare, următoarele 2000 de imagini din directorul data
- ***validation_labels*** - reprezintă clasele fiecarei imagini din setul de validare (0 - dacă pacientul nu prezintă anomalie, 1 în caz contrar)
- ***imagini_test*** - reprezintă imaginile din setul de testare, ultimele 5149 de imagini din directorul data

III. Modele utilizate

Modelele pe care le-am abordat pentru realizarea acestui proiect au fost Multinomial Naïve-Bayes (MultinomialNB) și Rețele Neuronale Convoluționale (CNN).

IV. Citirea și generarea fișierului CSV

- Pentru a extrage numele fișierelor corespunzătoare pozelor am folosit funcția os.listdir() din biblioteca OS.
- Pentru a citi imaginile am implementat 3 funcții, în cadrul cărora am folosit biblioteca PIL pentru citirea propriu-zisă, după care am convertit toate pozele la grayscale (img.convert("L")) și le-am transformat din vectori tridimensionali în vectori unidimensionali cu funcția .flatten() pentru Modelul Multinomial Naïve-Bayes, însă pentru CNN am lăsat pozele în formatul lor original (vector tridimensional). Vectorii de imagini vor fi returnați ca np.array.
- Pentru a citi etichetele am implementat o funcție separată care va returna un np.array conținând valorile claselor, indicii corespunzând pozelor descrise.
- Pentru a genera fișierul CSV am reținut toate label-urile prezise într-un vector și am aplicat formatul corespunzător, după care am utilizat funcția np.savetxt() incluzând și header-ul necesar, fără comentariu.

1) Multinomial Naïve-Bayes

Introducere :

Modelul Multinomial Naïve-Bayes se bazează pe teoria probabilităţilor şi se foloseste de o distribuție multinomială pentru a calcula probabilitatea ca o imagine să aparțină unei anumite clase.

Modelul presupune că fiecare imagine este independentă și că toate imaginile sunt la fel de relevante, ceea ce nu va conduce la rezultate spectaculoase de acuratețe, de aici și numele "naiv".

Descrierea modelului

Pentru a optimiza modelul am folosit o segmentare a datelor în 4 intervale, în cadrul cărora am considerat valorile maxime ale pixelilor 224, ceea ce a condus la scorurile următoare :

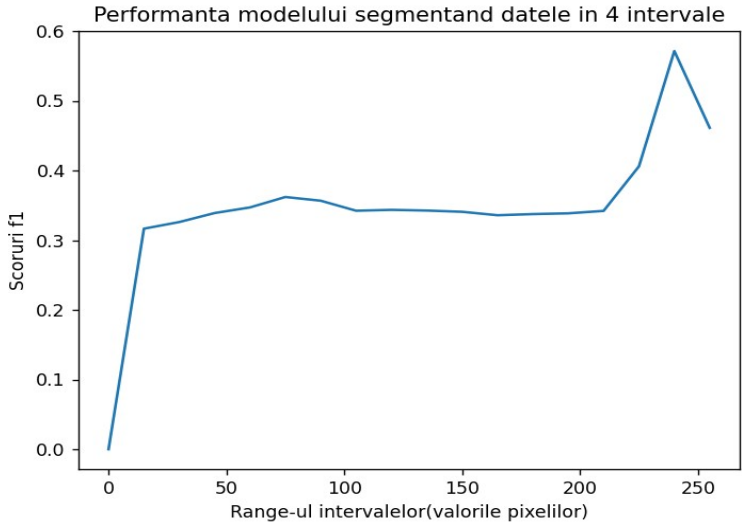
- ***Accuracy :*** 0.723
- ***F1_score :*** 0.4068522483940043

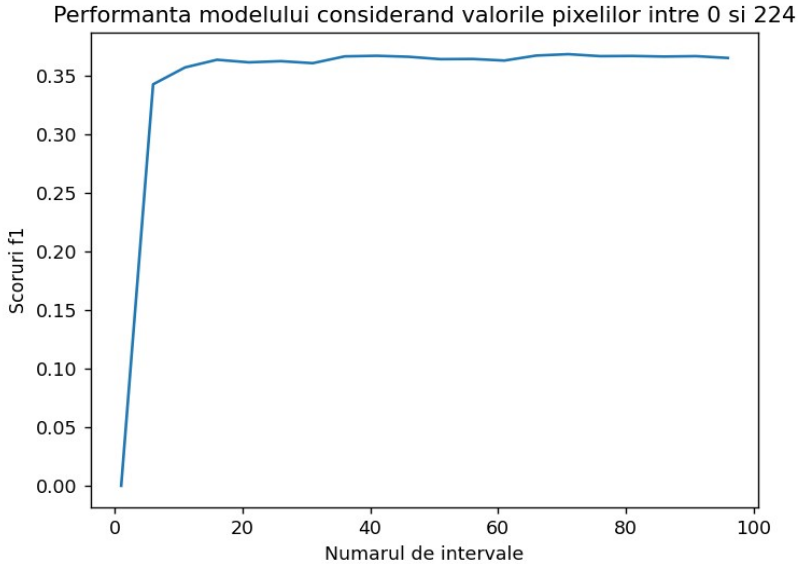
Utilizând abordarea din laborator, am creat două funcţii

- ***generare_intervale*** folosește funcția np.linspace(), care va segmenta intervalul [min, max] în num_intervale, în cazul meu output-ul fiind [0. 74.66666667 149.33333333 224.]

- ***imparte_in_intervale*** folosește funcția np.digitize(), care va asigna fiecărui pixel din vectorii de imagini o clasă, numărul claselor fiind numărul de intervale generate - 1, în cazul meu clasele vor fi 0, 1, 2.

Rezultatele abordării sunt ilustrate în următoarele grafice :





Graficele demonstrează faptul că limita superioară a intervalului optim pentru valorilor pixelilor ar fi fost cuprinsă între 230 și 240, cu aproximație.

Scorurile pentru această valoare sunt :

- ***Accuracy :*** 0.874
- ***F1_score :*** 0.5728813559322035

De asemenea, am încercat ***normalizatea datelor*** în cele 3 forme prezentate în cadrul laboratorului (standardizarea, normalizarea L1 si normalizarea L2), însa acuratețea nu a fost satisfăcătoare.

În plus, am dorit să folosesc ***Data Augmentation***, incluzând în setul datelor de antrenare ***imaginile rotite*** în toate cele 4 moduri (0°, 90°, 180°, 270°) și imaginile blurate folosind ***GaussianBlur()*** din biblioteca PIL, însă acestea au condus la overfitting, și, așadar, la rezultate nesatisfăcătoare.

Varianta finală, cea în care am folosit 4 intervale și valoarea maximă a pixelilor 224 a generat următoarea matrice de confuzie :

[[1256. 468.]

[86. 190.]]

2) Rețea Neuronală Convoluțională (CNN)

Introducere

O Rețea Neuronală Convoluțională (CNN) funcționează prin aplicarea anumitor filtre (kernel-uri) peste imagine, acestea fiind glisate peste întreaga suprafață a imaginii. Acestea sunt matrici de mici dimensiuni cu ajutorul cărora modelul poate detecta caracteristici specifice (linii, forme, margini ale obiectelor, etc).

Descrierea modelului final

```
CNN_model = Sequential()
CNN_model.add(Conv2D(16, (3,3), activation = 'relu', input_shape = (224, 224, 3)))
CNN_model.add(BatchNormalization())

CNN_model.add(Lambda(lambda x : x / 255))
CNN_model.add(MaxPooling2D((2, 2)))

CNN_model.add(Conv2D(32, (3,3), activation = 'relu'))
CNN_model.add(BatchNormalization())
CNN_model.add(MaxPooling2D((2, 2)))

CNN_model.add(Conv2D(64, (3,3), activation = 'relu'))
CNN_model.add(BatchNormalization())
CNN_model.add(MaxPooling2D((2, 2)))

CNN_model.add(Conv2D(128, (3,3), activation = 'relu'))
CNN_model.add(BatchNormalization())
CNN_model.add(MaxPooling2D((2, 2)))

CNN_model.add(Flatten())

CNN_model.add(Dense(256, activation = 'relu'))
CNN_model.add(Dense(128, activation = 'relu'))
CNN_model.add(Dense(64, activation = 'relu'))
CNN_model.add(Dense(32, activation = 'relu'))
CNN_model.add(Dense(1, activation = 'sigmoid')
```

Sumarul modelului

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_50 (Conv2D)          (None, 222, 222, 16)      448

 batch_normalization_50 (Bat  (None, 222, 222, 16)     64
 chNormalization)

 lambda_8 (Lambda)           (None, 222, 222, 16)      0

 max_pooling2d_31 (MaxPoolin  (None, 111, 111, 16)     0
 g2D)

 conv2d_51 (Conv2D)          (None, 109, 109, 32)      4640

 batch_normalization_51 (Bat  (None, 109, 109, 32)     128
 chNormalization)

 max_pooling2d_32 (MaxPoolin  (None, 54, 54, 32)       0
 g2D)

 conv2d_52 (Conv2D)          (None, 52, 52, 64)        18496

 batch_normalization_52 (Bat  (None, 52, 52, 64)       256
 chNormalization)

 max_pooling2d_33 (MaxPoolin  (None, 26, 26, 64)       0
 g2D)

 conv2d_53 (Conv2D)          (None, 24, 24, 128)       73856

 batch_normalization_53 (Bat  (None, 24, 24, 128)      512
 chNormalization)

 max_pooling2d_34 (MaxPoolin  (None, 12, 12, 128)      0
 g2D)

 flatten_8 (Flatten)         (None, 18432)             0

 dense_22 (Dense)            (None, 256)               4718848

 dense_23 (Dense)            (None, 128)               32896

 dense_24 (Dense)            (None, 64)                8256

 dense_25 (Dense)            (None, 32)                2080

 dense_26 (Dense)            (None, 1)                 33

=================================================================
Total params: 4,860,513
Trainable params: 4,860,033
Non-trainable params: 480
_________________________________________________________________
```

Compilarea modelului :

- Optimizatorul folosit a fost ***Adaptive Moment Estimation (ADAM)***, pentru a îmbunătăți performanța generală și deoarece necesită puțină memorie, cât și computații.
- Pentru funcția loss am folosit ***BinaryCrossentropy***, ea fiind folosită des în probleme de clasificare binară (exact cazul nostru). Aceasta asigură faptul că loss-ul va scădea treptat între epoci, deci și o performanță mai bună a modelului.
- Metrica folosită va fi ***'accuracy'***, deoarece rezultatul relevant pentru noi este proporția în care modelul a reușit să clasifice corect imaginile.

Antrenarea modelului :

Am antrenat modelul în ***20 de epoci***, în ***batch-uri de 64 de imagini***. Istoricul epocilor este următorul :

```
Epoch 1/20
235/235 [==============================] - 51s 148ms/step - loss: 0.4751 - accuracy: 0.8431
Epoch 2/20
235/235 [==============================] - 35s 147ms/step - loss: 0.3277 - accuracy: 0.8616
Epoch 3/20
235/235 [==============================] - 35s 151ms/step - loss: 0.3043 - accuracy: 0.8681
Epoch 4/20
235/235 [==============================] - 36s 153ms/step - loss: 0.2889 - accuracy: 0.8753
Epoch 5/20
235/235 [==============================] - 36s 152ms/step - loss: 0.2731 - accuracy: 0.8867
Epoch 6/20
235/235 [==============================] - 36s 152ms/step - loss: 0.2575 - accuracy: 0.8957
Epoch 7/20
235/235 [==============================] - 36s 152ms/step - loss: 0.2448 - accuracy: 0.8992
Epoch 8/20
235/235 [==============================] - 36s 152ms/step - loss: 0.2373 - accuracy: 0.9023
Epoch 9/20
235/235 [==============================] - 36s 152ms/step - loss: 0.2244 - accuracy: 0.9095
Epoch 10/20
235/235 [==============================] - 36s 152ms/step - loss: 0.2203 - accuracy: 0.9118
Epoch 11/20
235/235 [==============================] - 36s 152ms/step - loss: 0.2105 - accuracy: 0.9155
Epoch 12/20
235/235 [==============================] - 36s 152ms/step - loss: 0.2007 - accuracy: 0.9185
Epoch 13/20
235/235 [==============================] - 36s 152ms/step - loss: 0.1899 - accuracy: 0.9233
Epoch 14/20
235/235 [==============================] - 36s 152ms/step - loss: 0.1812 - accuracy: 0.9279
Epoch 15/20
235/235 [==============================] - 36s 152ms/step - loss: 0.1731 - accuracy: 0.9287
Epoch 16/20
235/235 [==============================] - 36s 152ms/step - loss: 0.1583 - accuracy: 0.9348
Epoch 17/20
235/235 [==============================] - 36s 152ms/step - loss: 0.1498 - accuracy: 0.9408
Epoch 18/20
235/235 [==============================] - 36s 152ms/step - loss: 0.1335 - accuracy: 0.9472
Epoch 19/20
235/235 [==============================] - 36s 152ms/step - loss: 0.1263 - accuracy: 0.9483
Epoch 20/20
235/235 [==============================] - 36s 152ms/step - loss: 0.1095 - accuracy: 0.9547
```

Pentru acest model am obținut scorurile :

- ***Loss :*** 0.4293023347854614
- ***Accuracy :*** 0.8980000019073486
- ***F1_score :*** 0.5919662136177521

Layer-ele folosite:

- ***Conv2D*** – Acest strat realizează aplicarea filtrelor. L-am aplicat în mod repetat, crescând progresiv numărul de filtre (16, 32, 64, 128), însa dimensiunea filtrelor a rămas neschimbată (3x3) pentru a selecta cel mai bine trăsăturile imaginii. Stride-ul a ramas cel implicit de 1, întrucât am dorit ca filtrele să fie aplicate pe toate porțiunile imaginii, iar funcția de activare este de tip 'relu', întrucât este un optimizator simplu și rapid, obținând rezultate mai bune decât 'sigmoid'. Primul layer va primi ca input_shape dimensiunea unei poze (224 x 224 x 3).

- ***MaxPooling2D*** - Imaginile primite de acest strat sunt împărțite în cazul meu în regiuni de (2x2), astfel păstrând doar informația relevantă din ele. Stratul va returna pentru fiecare regiune de 2x2 valoarea maximă a pixelilor din acea regiune.

- ***Lambda*** - Acest strat are rolul de a normaliza datele, modelul lucrând mult mai bine dacă valorile pixelilor sunt valori reale, între 0 și 1.

- ***Flatten*** - Stratul Flatten are rolul de a transforma informația dată de layer-ele Conv2D și MaxPooling2D într-un vector unidimensional, care poate fi dat ca input unei rețele conectate complet.

- ***Dense*** - Acest strat va efectua o transformare liniară a datelor și va lega fiecare neuron din stratul precedent de fiecare neuron curent, în cazul meu aplicat progresiv (256, 128, 64, 32, 1). Această formare de legături ne ajută să descoperim relații non-liniare între caracteristicile datelor.

- ***BatchNormalization*** - Stratul BatchNormalization are rolul de a sabiliza procesul de antrenare și de a reduce probabiliatea overfitting-ului. Per total, acesta va îmbunătăți acuratețea modelului.

O versiune mai slabă a modelului

```
CNN_model = Sequential()

CNN_model.add(Conv2D(16, (3,3), 1, activation = 'relu', input_shape = (224, 224, 3)))
CNN_model.add(MaxPooling2D((2, 2)))

CNN_model.add(Lambda(lambda x : x / 255))

CNN_model.add(Conv2D(32, (3,3), 1, activation = 'relu'))
CNN_model.add(MaxPooling2D((2, 2)))

CNN_model.add(Conv2D(64, (3,3), 1, activation = 'relu'))
CNN_model.add(MaxPooling2D((2, 2)))

CNN_model.add(Conv2D(128, (3,3), 1, activation = 'relu'))
CNN_model.add(MaxPooling2D((2, 2)))

CNN_model.add(Flatten())
CNN_model.add(Dense(64, activation = 'relu'))
CNN_model.add(Dense(1, activation = 'sigmoid'))
```

Pentru acest model am obținut scorurile :

- ***Loss :*** 0.5183745630792702
- ***Accuracy :*** 0.8634765300043217
- ***F1_score :*** 0.5584675359120008

De asemenea, am încercat adăugarea de layere 'Dropout' însă aceasta a rezultat în underfitting, aşadar la un rezultat mai slab.

Varianta finală a generat următoarea matrice de confuzie :

[[1622. 102.]

[130. 146.]]

<u>Graficele pentru loss si acurateţe :</u>