Brain Anomaly Detection

"The task is to discriminate between two classes of brain CT scans, one that contains anomalies (label 1) and one that is normal (class 0). Each sample is a grayscale image of 224x224 pixels."

Pentru rezolvarea acestei probleme, am antrenat doua modele diferite; CNN Classifier si Naive Bayes Classifier. Tot proiectul a fost implementat in Google Colab, asa ca, pentru inceput, mi-am preluat din Google Drive zip-ul unde se aflau toate pozele si fisierele cu datele de antrenare si de testare.

1. Preprocesare

Preprocesarea datelor a fost comuna pentru ambele modele (in cea mai mare parte): mi-am salvat path-ul de unde preiau datele de testare si antrenare, mi-am creat listele de train_labels, validation_labels si test_labels, folosind functia **np.getextfrom**, din libraria numpy, iar apoi am citit toate pozele din folderul "data". Pentru citirea pozelor (care erau fisiere de tip **.png**), am folosit mai multe functii:

- am folosit modulul os pentru a accesa funcții care gestionează sistemul de operare și modulul glob pentru a găsi toate fișierele PNG dintr-un anumit director dat.
- os.path.join() : am folosit aceasta functie pentru a citi eficient fiecare imagine. Din cauza faptului ca imaginile aveau in denumire numarul imaginii, iar toate numele imaginilor erau diferite, os.path.join m-a ajutat sa citesc toate fisierele din data_path unde am extensia .png, deoarece cu asta se ocupa el: pot îmbina mai multe părți ale unei căi într-una singură, în loc să codific manual fiecare nume de cale.
- am folosit funcția sorted() pentru a sorta fișierele găsite în funcție de cheia specificată.
- am specificat o cheie pentru a sorta fișierele în funcție de numarul din numele fișierului PNG. Pentru a face acest lucru, ne-am folosit de functia lambda x: int(os.path.basename(x).split(".")[0]), care ia fiecare fișier din lista, extrage numărul din numele fișierului PNG (folosind os.path.basename() pentru a obține numele de fișier si split() pentru a împărți numele fisierului în funcție de punct), apoi îl convertestim într-un integer folosind int(). Fișierele PNG sunt sortate astfel în ordinea numerelor din numele lor

In final, lista sortată de fișiere PNG este atribuită variabilei 'png_files'.

Am vrut ca primele 17000 de poze sa fie salvate in train_images, deoarece, conform cerintei, acestea au fost destinate pentru a ne antrena modelele cu ele, iar in test_images, le am salvat

pe cele incepand cu indicele 17000 si pana la final, acestea reprezentand imaginile cu care vom compara ceea ce am antrenat noi.

2. Antrenarea modelelor

★ Naive Bayes Classifier

Clasificatorul Naive Bayes este un algoritm de învățare automată supravegheat, care este utilizat pentru sarcini de clasificare, cum ar fi clasificarea textului.

Pentru inceput, trebuie mentionat faptul ca in preprocesarea datelor pentru acest clasificator, ne-am folosit si de **StandardScaler()**. StandardScaler este o tehnică comună de preprocesare a datelor în invatarea automata și în analiza de date. Acesta este utilizat pentru a redimensiona și transforma variabilele în așa fel încât acestea să aibă o medie de zero și o deviație standard de 1. Aceasta ajută la eliminarea variației între variabile, permitandu-le să fie comparate în mod egal în cadrul unui model.

Procesul de scalare standard este realizat prin următorii pași:

- ➢ calculam media și deviația standard a fiecărei variabile din setul de date.
- ➢ folosim aceste statistici pentru a transforma fiecare valoare a fiecarei variabile astfel încât să aibă o medie de zero și o deviație standard de 1.
- ➢ variabilele scalate astfel sunt apoi folosite pentru a antrena modele de învățare automată sau pentru a efectua analiza de date.

De asemenea, pentru a ne preprocesa modelul, ne-am folosit de functia **GaussianNB()**.

```
n_classes = len(np.unique(train_labels))
class_counts = np.bincount(train_labels)
class_priors = class_counts / len(train_labels)
modelNB =GaussianNB(priors = class_priors, var_smoothing = 0.001)
```

Secventa de cod de mai sus are rolul de a pregăti datele și de a inițializa un model de învățare automată de tipul Naive Bayes.

- ➢ n_classes = len(np.unique(train_labels)): calculează numarul de clase unice din setul de antrenament, folosind funcția np.unique() pentru a găsi toate valorile unice din lista train_labels, iar apoi funcția len() pentru a număra numarul de valori unice. Variabila n_classes stochează acest numar
- ➢ class_counts = np.bincount(train_labels): calculează numărul de exemple din fiecare clasa din train_labels. Folosind funcția np.bincount(), calculam numarul de aparitii al

fiecarei clase din lista train_labels. Rezultatele sunt stocate în variabila class_counts, unde fiecare element corespunde cu numărul de exemple dintr-o anumita clasa

- ➢ class_priors = class_counts / len(train_labels): calculam probabiltatile fiecărei clase, adică probabilitatea de a obține un exemplu dintr-o anumită clasă înainte de a vedea datele. Acestea sunt calculate prin impartirea numarului de exemple din fiecare clasă din setul de antrenament (class_counts) la numărul total de exemple din setul de antrenament (len(train_labels)). Variabila class_priors stochează aceste probabilitati
- ➢ modelNB =GaussianNB(priors = class_priors, var_smoothing = 0.001): initalizam un model de clasificare Naive Bayes de tipul GaussianNB, folosind valoarea probabilităților calculate mai devreme pentru fiecare clasa(class_priors) și o valoare mică de largire(var_smoothing) pentru a evita probabilitățile de zero atunci când sunt calculate distribuțiile Gaussiene în cadrul algoritmului Naive Bayes.

În concluzie, această secvență de cod calculează probabilitățile claselor și initalizeaza un model de tipul GaussianNB pentru a fi antrenat cu datele din setul de antrenament.

In continuare, ne vom antrena modelul pe batch-uri. Am ales din 100 in 100 de imagini, deoarece pentru mai multe imagini, Google Colab avea crash.

```
batch_size = 100
for i in range(0, len(train_images), batch_size):
  image_batch = []
  for file in train_images[i:i+batch_size]:
    image = cv2.imread(file)
    image = image.flatten()
    image_batch.append(image)
  norm_image_batch = standardScaler.fit_transform(image_batch)
     modelNB.partial_fit(norm_image_batch,train_labels[i:i+batch_size],
classes=[0, 1])
```

Incepem prin a parcurge setul de date de antrenament și il impartim în loturi de dimensiunea specificata mai sus. În fiecare iteratie, se citește imaginea și pixelii acesteia. Acestia sunt aplatizati într-un singur vector și sunt adaugați la batch-ul curent. După aceea, batch-ul de imagini este normalizat folosind standardScaler-ul creat in preprocesare. Acesta realizeaza scalarea datelor astfel încât media valorilor să fie 0 și deviația standard să fie 1.

În final, modelul este actualizat cu lotul curent de imagini și cu etichetele corespunzătoare. În acest caz, se folosește metoda partial_fit() pentru a actualiza modelul în mod incremental, ceea ce permite antrenarea pe seturi de date de dimensiuni mari și cu resurse limitate.

Astfel, prin parcurgerea întregului set de date de antrenament în batch-uri mai mici și prin actualizarea incrementală a modelului în fiecare iterație, se obține un model capabil să clasifice imagini noi cu 0 sau 1.

```
y_pred = []
batch_size = 50
for i in range(0, len(test_images)+1, batch_size):
  image_batch = []
  for file in test_images[i:i+batch_size]:
    image = cv2.imread(file)
    image = image.flatten()  # flatten the pixel array to a 1D vector
    image_batch.append(image)
  image_batch = standardScaler.fit_transform(image_batch)
  partialPredict = modelNB.predict(image_batch)
  y_pred.append(partialPredict)

y_pred = np.concatenate(y_pred, axis=0)
```

În primul rand, initializam o listă goală y_pred care va fi utilizată pentru a stoca predicțiile modelului pe setul de date de testare.

Apoi, stabilim dimensiunea batch-ului(batch size), care reprezintă numărul de exemple de testare pe care le vom utiliza pentru a face predicții în fiecare iterație. Am ales ca batch-ul sa aiba dimensiunea de 50, tot din motivul crash-ului.

În continuare, parcurgem setul de date de testare și se împarte în batch-uri de dimensiunea specificata mai sus. În fiecare iterație, citim imaginea și pixelii acesteia. Aceatia sunt aplatizați într-un singur vector și sunt adaugați la batch-ul curent.

Dupa aceea, exact ca mai devreme, batch-ul de imagini este normalizat folosind standardScaler. În continuare, se face o predicție parțială pe lotul de imagini folosind metoda predict() a modelului Naive Bayes antrenat anterior. Această metodă returnează o matrice de predictii pentru batch-ul curent de imagini. Predictiile partiale sunt apoi adaugate la lista y_pred.

La final, lista y_pred este concatenata folosind funcția concatenate() și se obține o matrice cu predicțiile finale pentru toate exemplele de testare.

```
ids=np.genfromtxt('/content/data/sample_submission.txt', delimiter=',',
dtype='int')[1:][:,0]
```

Această secvență de cod încarcă un fișier CSV cu o listă de identificatori unici pentru setul de date pe care se dorește efectuarea unei predicții și salvează acești identificatori într-un vector NumPy numit ids.
Mai întâi, se folosește funcția genfromtxt() pentru a încarca fisierul. Parametrul delimiter=',' specifică faptul că fișierul este un fișier CSV și că valorile din el sunt separate prin virgulă. Parametrul dtype='int' specifică faptul că valorile din fișier sunt de tip întreg. De asemenea, se selectează doar prima coloană (care conține identificatorii) din restul liniilor din fișier folosind [0].
În final, rezultatul este un vector care conține identificatorii unici ai exemplelor din setul de date pe care se dorește efectuarea unei predicții.

La finalul codului, am creat fisierul CSV in care am salvat toate predicțiile, insotite de id-ul fiecarei imagini si am folosit functiile din sklearn.metrics pentru a calcula si afisa urmatoarele: acuratetea, matricea de confuzie, precizia, recall_score (ne va ajuta pentru f1_score) si f1_score, scorul care imi arata si pe Kaggle cat de bine am antrenat modelul.

★ CNN Classifier

O rețea neuronală convoluțională, sau pe scurt CNN, este un tip de clasificator, care excelează în rezolvarea acestei probleme. Un CNN este o rețea neuronală: un algoritm folosit pentru a recunoaște modele în date.
Inainte de a prezenta modelul efectiv, trebuie mentionat si aici cu ce difera preprocesarea datelor fata de clasificatorul Naive Bayes.

```
n_classes = len(np.unique(train_labels))
class_counts = np.bincount(train_labels)
class_priors = class_counts / len(train_labels)
```

In aceasta secventa de cod am calculat distribuția claselor din setul de date și prioritățile acestora pentru a fi folosite într-un model de învățare supervizată.
În primul rând, am calculat numărul de clase diferite din variabila train_labels prin apelarea funcției np.unique(train_labels). Rezultatul este stocat în variabila n_classes.
Apoi, am folosit funcția np.bincount(train_labels) pentru a calcula numarul de exemple din fiecare clasă în variabila train_labels. Această funcție numara aparițiile fiecarei valori întregi dintr-un vector și le stochează într-un nou vector.
În final, se calculează vectorul class_priors care conține probabilitatea fiecarei clase în setul de date prin împărțirea numărului de exemple din fiecare clasă cu numărul total de exemple din setul de date (len(train_labels)).

```
train_images = []
for path in png_files[:17000]:
    img = cv2.imread(path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = np.resize(img, (224,224,3))
    train_images.append(img)
train_images,train_labels=shuffle(train_images,np.concatenate((train_la
bels, validation_labels),axis=0))
```

In aceasta secventa de cod, încarcam imaginile de antrenare pentru modelul nostru si le pregatim pentru antrenare prin redimensionare, conversie în format RGB și amestecarea lor cu setul de date de validare.

În primul rând, se initializeaza o listă goala train_images pentru a stoca imaginile de antrenare încărcate. Apoi, se folosește un loop for pentru a parcurge primele 17000 de fișiere PNG din lista png_files. Fiecare imagine PNG este încărcată folosind funcția cv2.imread(). Această funcție citește imaginea de la calea specificată și o converteste folosind functia cv2.cvtColor(). În continuare, fiecare imagine este redimensionată la dimensiunea de 224x224x3 folosind funcția np.resize(), pentru a se potrivi dimensiunii de intrare a modelului. În final, lista train_images este amestecată impreuna cu validation_labels și etichetele corespunzatoare sunt stocate în variabila train_labels folosind funcția shuffle(). Această funcție amestecă elementele din lista train_images și din vectorul de etichete np.concatenate((train_labels, validation_labels),axis=0) (care conține etichetele atat pentru setul de date de antrenare, cât și pentru cel de validare), astfel încât elementele din cele două seturi să nu fie grupate în mod accidental în timpul antrenării. Am folosit shuffle, deoarece, initial, programul nu imi stoca imaginile in ordine si astfel aveam erori, deoarece indicii nu erau in ordine, deci nu le puteam parcurge pe toate pentru a-mi antrena modelul.

```
# Load test images
test_images = []
for path in png_files[17000:]:
    img = cv2.imread(path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = np.resize(img, (224,224,3))
    test_images.append(img)
```

Aceasta secventa de cod încarcă imaginile de testare și le pregătește pentru evaluarea modelului meu. In plus, am initializat o listă goală test_images pentru a stoca imaginile de testare încărcate.

Folosim un loop for pentru a parcurge toate fișierele PNG din lista png_files, incepand cu imaginea de pe pozitia 17000 si pana la final. Fiecare imagine PNG este încărcată folosind funcția cv2.imread(), exact cum am procedat si pentru train_images. Fiecare imagine

procesată este apoi adăugată la lista test_images. Aceste imagini vor fi utilizate pentru a evalua performanța modelului după ce acesta a fost antrenat pe setul de date de antrenare.

```
model = tf.keras.Sequential([
     tf.keras.layers.Conv2D(16, (3,3), activation = 'relu', input_shape
= (224, 224,3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation= 'relu'),
    tf.keras.layers.Dense(1, activation = 'sigmoid')
])
```

In aceasta secventa de cod am definit modelul de rețea neuronală convoluțională (CNN) în TensorFlow.

Definim un obiect de tip Sequential din biblioteca Keras de la TensorFlow, care permite definirea modelului strat cu strat. Modelul are următoarele straturi:

- Conv2D: un strat de convoluție 2D cu 16 filtre, fiecare de dimensiune (3,3), cu funcția de activare ReLU. Parametrul input_shape specifică dimensiunea imaginilor de intrare la stratul de intrare, în acest caz fiind (224, 224, 3), ceea ce înseamnă că imaginile de intrare au o rezoluție de 224x224 pixeli și 3 canale de culoare (RGB).
- MaxPooling2D: un strat de reducere a dimensiunii cu o fereastră de 2x2 și o mișcare de 2x2, ceea ce înseamnă că imaginea de intrare este împărțită în zone de 2x2 pixeli și se extrage valoarea maximă din fiecare zonă, astfel încât dimensiunea imaginii se reduce la jumătate.
- Flatten: un strat care transformă o matrice 2D de pixeli într-un vector 1D lung, pentru a fi conectat la straturi dense.
- Dense: un strat dens (total conectat) cu 128 de neuroni și funcția de activare ReLU.
- Dense: un strat dens cu un singur neuron și funcția de activare sigmoid, care este folosit pentru clasificarea binară (exemplu: un obiect este prezent în imagine sau nu).

Aceste straturi definesc modelul de rețea neuronală convoluțională care poate fi compilat și antrenat pentru clasificarea imaginilor binare.

```
model.compile(loss = tf.keras.losses.binary_crossentropy, optimizer=
tf.keras.optimizers.Adam(), metrics= ["accuracy"])
train_images = np.array(train_images)
train_labels = np.array(train_labels)
history = model.fit(x=train_images, y=train_labels, epochs = 5)
print(np.shape(train_images), np.shape(train_labels))
```

Aceasta secventa de cod arată procesul de compilare și antrenare a unui model CNN.

În primul rând, modelul este definit folosind funcția tf.keras.Sequential(), care creează o stivă liniară de straturi. În acest caz, modelul constă dintr-un strat convoluțional, un strat de pooling maxim, un strat de aplatizare, un strat dens cu 128 de noduri și un strat de ieșire cu un singur nod și o funcție de activare sigmoid.
După definirea arhitecturii modelului, acesta este compilat folosind funcția model.compile(). Funcția de pierdere este setată la binary_crossentropy deoarece aceasta este o problemă de clasificare binară, iar optimizatorul este setat la optimizatorul Adam cu parametrii impliciti. Valoarea pentru evaluare este setată la acuratețe.
Apoi, datele train_images și train_labels sunt convertite intr-o matrice, folosind funcția np.array(). Modelul este apoi antrenat folosind funcția model.fit() cu train_images și train_labels ca intrări și epochs = 5 pentru a indica numarul de epoci de antrenament.
În cele din urmă, formele train_images și train_labels sunt tiparite folosind np.shape() pentru a verifica dacă se potrivesc cu dimensiunile impuse.

```
test_images = np.array(test_images)
y_pred = (model.predict(test_images) > 0.5).astype("int32")
```

În aceasta secventa de cod, imaginile_test sunt convertite într-o matrice folosind funcția np.array().
Apoi, funcția model.predict() face predictii asupra lui test_images. Modelul calculeaza o probabilitate pentru fiecare imagine, indicand probabilitatea ca imaginea sa aparțina clasei pozitive.
Pentru a obține predicțiile binare, se aplică un prag de 0,5 probabilităților prezise. Dacă probabilitatea este mai mare de 0,5, imaginea este clasificată ca clasa pozitivă, iar elementul corespunzător din tabloul y_pred este setat la 1; în caz contrar, este clasificată ca clasa negativă, iar elementul corespunzător este setat la 0. Acest lucru se realizează prin utilizarea metodei astype() pentru a converti tabloul boolean obținut din (model.predict(test_images) > 0,5) într-un tablou întreg. cu valorile 0 si 1.
Prin urmare, la sfarsitul segmentului de cod, y_pred este o matrice care conține etichetele prezise pentru datele test_images, pe care, in urmatoarea linie de cod o vom concatena cu functia np.concatenate(), pentru a ne crea o singura lista in jurul axei 0 (distribuirea datelor pe verticala).

In final, exact ca la clasificatorul Naive Bayes, am creat fisierul CSV unde am salvat predictiile, am calculat acuratetea, precizia, recall_score si f1_score si le-am afisat.