Hârnagea Andrei-Alexandru
Grupa 242

Brain Anomaly Detection

Detect anomalies in CT scans of the brain

1. Introducere & Descrierea poiectului

Acest proiect a constat în clasificarea de imagini monocrome a tomografiilor la creier. Acestea puteau fi încadrate în două clase: dacă este o anomalie sau nu. Fiecare imagine are dimensiunea de 224 x 224 pixeli.

2. Setul de date

Pentru a clasifica imaginile, am primit ca și date:

- 15000 poze de antrenare, însoțite de label-urile lor
- 2000 poze de validare, însoțite de lab-urile lor
- 5149 poze de testare (fără label)

3. Modele folosite

I. ***Naïve Bayes –*** acesta este un algortim de învățare automată, bazat pe teorema lui Bayes. Algoritmul este "naïve" deoarece face o presupunere că toate caracteristicile sunt independente între ele. Am ales acest algoritm datorită eficienței și facilității de implementare ca și primă metodă de abordare.

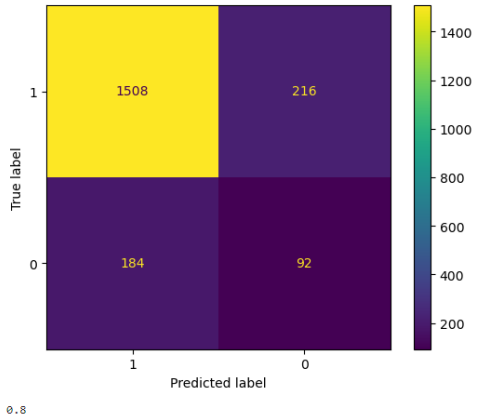
În ce privește prelucrarea datelor, am aplicat imaginilor un flatten la citire, după ce le-am convertit la grayscale, și le-am adăugat unui numpy array.

```
poze_train = []
for index in list_id_train:
    image = cv2.imread(f'/kaggle/input/unibuc-brain-ad/data/data/'+index+'.png', cv2.IMREAD_GRAYSCALE)
    poze_train.append((image).flatten())
poze_train = np.array(poze_train)

poze_sample = []
for index in list_id_sample:
    image = cv2.imread(f'/kaggle/input/unibuc-brain-ad/data/data/{index}.png', cv2.IMREAD_GRAYSCALE)
    poze_sample.append((image).flatten())
poze_sample = np.array(poze_sample)

poze_validation = []
for index in list_id_validation:
    image = cv2.imread(f'/kaggle/input/unibuc-brain-ad/data/data/{index}.png', cv2.IMREAD_GRAYSCALE)
    poze_validation.append((image).flatten())
poze_validation = np.array(poze_validation)
```

Hârnagea Andrei-Alexandru
Grupa 242

Am folosit două variante ale algoritmului: **GaussianNB** și **BernoulliNB**.

i. **GaussianNB** - se bazează pe faptul că datele de intrare urmează o distribuție normală. Cu acest algoritm am obținu t un scor de **0.34875**, și o acuratețe de 0.8

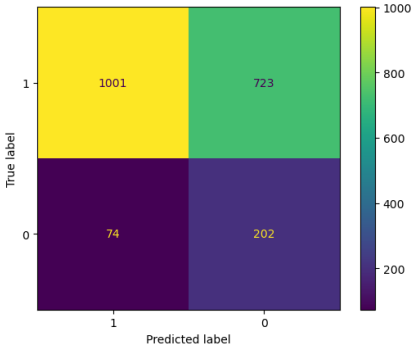


0.8

```
clf = GaussianNB()
clf.fit(poze_train, list_class_train)
y_pred = clf.predict(poze_validation)
```

```
Precision: 0.2987012987012987
Recall: 0.3333333333333333
```

ii. **BernoulliNB** – folosit în special pentru clasificarea datelor binare. Algoritmul presupune că toate caracteristicile sunt independente și că fiecare caracteristică este descrisă de o distribuție Bernoulli. Cu acest algoritm am obținut un scor de **0.32945,** și o acuratețe de 0.60.



0.6015

```
clf = BernoulliNB()
clf.fit(poze_train, list_class_train)
y_pred = clf.predict(poze_validation)
```

```
Precision: 0.21837837837837837
Recall: 0.7318840579710145
```

II. ***Random Forest –*** este o metodă care se bazează pe construirea unui ansambu de arbori de decizie. Fiecare arbore din ansambu este construit pe baza unui subset random de date de antrenare.Astfel, fiecare arbore este construit într-un mod diferit. Overfitting-ul este evitat prin construirea cu date aleatorii.

Citirea datelor este identică cu cea de la Naïve Bayes.

Parametrii:

n_estimators – numarul de estimatori din pădure(copaci)

max_features – numărul maxim de caracteristici pe care îl poate avea fiecare copac

max_depth – adâncimea maximă a fiecărui arbore. (valoare mare = over-fitting, valoare mică – sub-învățare)

min_samples_split – numărul minim de exemple necesare pentru a realiza un split într-un nod.

(Valoarea mare = sub-învățare, valoare mică = modelul este sensibil la zgomot)

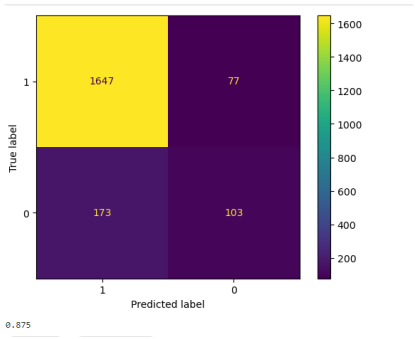
min_samples_leaf – numărul minim de exemple necesare pentru a formare un nod final într-un arbore (valoare mică = over-fitting, valoare mare – sub-învățare)

n_jobs = numărul de nuclee procesor utilizate

În prima încercare am creat următorul model:

```
rand_for = RandomForestClassifier(n_estimators = 50, max_features = 0.5,
                                  max_depth = 10, min_samples_split = 2,
                                  min_samples_leaf=1,n_jobs=-1)
rand_for = rand_for.fit(poze_train, list_class_train)
y_pred = rand_for.predict(poze_sample)
```

Am obținut un scor de **0.44881**, și o acuratețe de .0.875



În a doua încercare am modificat modelul astfel:

```
rand_for = RandomForestClassifier(n_estimators = 100, max_features = 0.5,
                                  max_depth = 20, min_samples_split = 2,
                                  min_samples_leaf=1,n_jobs=-1)
rand_for = rand_for.fit(poze_train, list_class_train)
y_pred = rand_for.predict(poze_sample)
```

Am obținut un scor de **0.4341**.

III. ***Convolutional Neural Network (CNN)*** – este o rețea neuronală făcută special pentru prelucrarea imaginilor. Acesta utilizează straturi de convoluție și pooling, care extrag și clasifică caracteristicile imaginilor. Pentru acest model am folosit keras din tenserflow. Citirea și prelucrarea datelor este asemănătoare celorlalte două modele. Am redimenstionat array-urile de poze pentru a putea fi preluate de model. Am decis să folosesc un model Sequential pentru că fiecare layer are exact un input și output. Fiecare model are un număr adjustabil de neuroni.

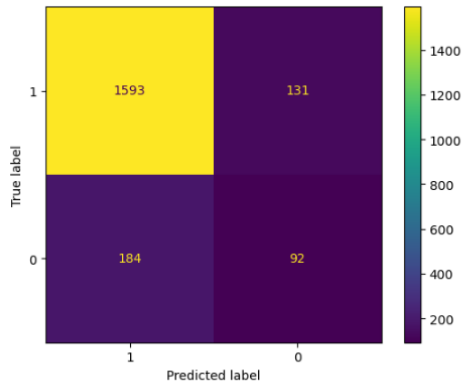
Primul model de CNN arată astfel:

Hârnagea Andrei-Alexandru
Grupa 242

```
model = Sequential()

model.add(Conv2D(256,(3,3), activation='relu', input_shape=(224,224,1)))
model.add(MaxPool2D(2,2))

model.add(Conv2D(64,(3,3), activation='relu', input_shape=(224,224,1)))
model.add(MaxPool2D(2,2))

model.add(Flatten())


model.add(Dense(64, activation = 'relu'))

model.add(Dense(1, activation = 'sigmoid'))

model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])


history = model.fit(poze_train,list_class_train, epochs = 5,
                    batch_size= batch_size, validation_data = (poze_validation,list_class_validation))

y_pred = model.predict(poze_sample)
```

Am obținut un scor de 0.32231 și o acuratețe de 0.8425.

```
Epoch 1/5
235/235 [==============================] - 40s 166ms/step - loss: 1.2447 - accuracy: 0.8461 - val_loss: 0.3418 - val_accuracy: 0.8620
Epoch 2/5
235/235 [==============================] - 38s 164ms/step - loss: 0.3457 - accuracy: 0.8507 - val_loss: 0.3491 - val_accuracy: 0.8620
Epoch 3/5
235/235 [==============================] - 38s 164ms/step - loss: 0.3104 - accuracy: 0.8509 - val_loss: 0.3463 - val_accuracy: 0.8625
Epoch 4/5
235/235 [==============================] - 38s 164ms/step - loss: 0.2695 - accuracy: 0.8793 - val_loss: 0.3814 - val_accuracy: 0.8615
Epoch 5/5
235/235 [==============================] - 38s 164ms/step - loss: 0.2269 - accuracy: 0.9063 - val_loss: 0.4153 - val_accuracy: 0.8425
63/63 [==============================] - 1s 19ms/step
```




```
0.8425
```

```
Precision: 0.31417624521072796
Recall: 0.2971014492753623
```

Hârnagea Andrei-Alexandru
Grupa 242

În a doua încercare, am mai adăugat mai multe elemente:

```
model = Sequential()

model.add(Conv2D(256,(3,3), activation='relu', input_shape=(224,224,1)))
model.add(MaxPool2D(2,2))
model.add(Dropout(0.2))

model.add(Conv2D(64,(3,3), activation='relu', input_shape=(224,224,1)))
model.add(MaxPool2D(2,2))
model.add(Dropout(0.2))

model.add(Conv2D(128,(3,3), activation='relu', input_shape=(224,224,1)))
model.add(MaxPool2D(2,2))
model.add(Dropout(0.2))

model.add(Conv2D(64,(3,3), activation='relu', input_shape=(224,224,1)))
model.add(MaxPool2D(2,2))
model.add(Dropout(0.2))

model.add(Conv2D(256,(3,3), activation='relu', input_shape=(224,224,1)))
model.add(MaxPool2D(2,2))
model.add(Dropout(0.5))

model.add(Conv2D(128,(3,3), activation='relu', input_shape=(224,224,1)))
model.add(MaxPool2D(2,2))
model.add(Dropout(0.5))

model.add(Flatten())


model.add(Dense(64, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation = 'sigmoid'))

model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])

history = model.fit(poze_train,list_class_train, epochs = 50, batch_size= batch_size, validation_data = (poze_validation,list_class_validation))
```

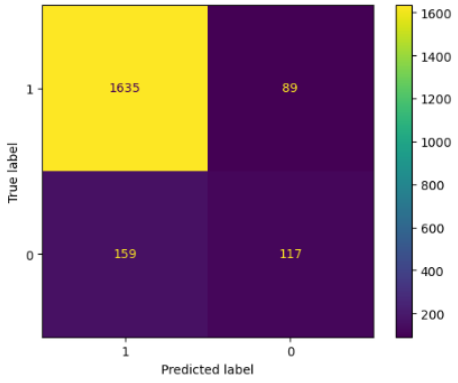
Am obținut astel un scor de 0.4975.

În al treilea model, am modificat codul astfel:

```
model = Sequential()

model.add(Conv2D(256,(3,3), activation='relu', input_shape=(224,224,1)))
model.add(MaxPool2D(2,2))


model.add(Conv2D(64,(3,3), activation='relu', input_shape=(224,224,1)))
model.add(MaxPool2D(2,2))
model.add(BatchNormalization())


model.add(Conv2D(128,(3,3), activation='relu', input_shape=(224,224,1)))
model.add(MaxPool2D(2,2))
model.add(BatchNormalization())


model.add(Conv2D(64,(3,3), activation='relu', input_shape=(224,224,1)))
model.add(MaxPool2D(2,2))
model.add(BatchNormalization())


model.add(Flatten())

model.add(Dense(64, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation = 'sigmoid'))

model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])

history = model.fit(poze_train,list_class_train, epochs = 15, batch_size= batch_size, validation_data = (poze_validation,list_class_validation))

y_pred = model.predict(poze_validation)
```

Hârnagea Andrei-Alexandru
Grupa 242

```
Epoch 1/15
235/235 [==============================] - 45s 177ms/step - loss: 0.3715 - accuracy: 0.8441 - val_loss: 0.3838 - val_accuracy: 0.8630
Epoch 2/15
235/235 [==============================] - 41s 175ms/step - loss: 0.3337 - accuracy: 0.8532 - val_loss: 0.3036 - val_accuracy: 0.8670
Epoch 3/15
235/235 [==============================] - 41s 175ms/step - loss: 0.3126 - accuracy: 0.8629 - val_loss: 0.3360 - val_accuracy: 0.8675
Epoch 4/15
235/235 [==============================] - 41s 175ms/step - loss: 0.3033 - accuracy: 0.8653 - val_loss: 0.6226 - val_accuracy: 0.8615
Epoch 5/15
235/235 [==============================] - 41s 174ms/step - loss: 0.2893 - accuracy: 0.8705 - val_loss: 0.3644 - val_accuracy: 0.8670
Epoch 6/15
235/235 [==============================] - 41s 174ms/step - loss: 0.2763 - accuracy: 0.8753 - val_loss: 0.3483 - val_accuracy: 0.8720
Epoch 7/15
235/235 [==============================] - 41s 175ms/step - loss: 0.2655 - accuracy: 0.8807 - val_loss: 0.3543 - val_accuracy: 0.8545
Epoch 8/15
235/235 [==============================] - 41s 174ms/step - loss: 0.2479 - accuracy: 0.8865 - val_loss: 0.2920 - val_accuracy: 0.8840
Epoch 9/15
235/235 [==============================] - 41s 175ms/step - loss: 0.2316 - accuracy: 0.8920 - val_loss: 0.3556 - val_accuracy: 0.8565
Epoch 10/15
235/235 [==============================] - 42s 180ms/step - loss: 0.2223 - accuracy: 0.9025 - val_loss: 0.3225 - val_accuracy: 0.8625
Epoch 11/15
235/235 [==============================] - 41s 175ms/step - loss: 0.2036 - accuracy: 0.9095 - val_loss: 0.3623 - val_accuracy: 0.8365
Epoch 12/15
235/235 [==============================] - 41s 175ms/step - loss: 0.1868 - accuracy: 0.9189 - val_loss: 0.3199 - val_accuracy: 0.8805
Epoch 13/15
235/235 [==============================] - 41s 175ms/step - loss: 0.1765 - accuracy: 0.9211 - val_loss: 0.4478 - val_accuracy: 0.8945
Epoch 14/15
235/235 [==============================] - 41s 175ms/step - loss: 0.1614 - accuracy: 0.9291 - val_loss: 0.3435 - val_accuracy: 0.8820
Epoch 15/15
235/235 [==============================] - 41s 174ms/step - loss: 0.1468 - accuracy: 0.9369 - val_loss: 0.4003 - val_accuracy: 0.8760
63/63 [==============================] - 1s 20ms/step
```




```
0.876
```

```
Precision: 0.4501347708894879
Recall: 0.605072463768116
```

Am obținut astfel un scor de 0.511 și o acuratețe de 0,876.

Am încercat o augumentare a datelor pe acest model, dar am obținut un scor mai mic:

```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
        rotation_range = 20,
        width_shift_range = 0.1,
        height_shift_range = 0.1,
        shear_range = 0.2,
        zoom_range = 0.2,
        horizontal_flip = True,
        vertical_flip = True,
        fill_mode = 'nearest'
)

poze_augumentate = []
labels_augumentate = []

for i in range(len(poze_train)):
    image = poze_train[i].reshape((224, 224, 1))
    label = list_class_train[i]

    for j in range(2):
        poza_aug = datagen.random_transform(image)
        poze_augumentate.append(poza_aug.flatten())
        labels_augumentate.append(label)

poze_train = np.concatenate((poze_train, poze_augumentate))
list_class_train = np.concatenate((list_class_train, labels_augumentate))
```

```
Scorul cu 0.45
63/63 [==============================] - 1s 20ms/step
0.3030682529743268
Scorul cu 0.5
63/63 [==============================] - 1s 19ms/step
0.30387794024157666
```

Hârnagea Andrei-Alexandru
Grupa 242

Explicare detaliată a modelelor CNN

Fiecare model este alcătuit din mai mulți neuroni. Primul model are 4 straturi, al doilea 8, iar ultimul 15 straturi. Prea puține straturi duce către sub-învățare, iar prea multe către over-fitting.

Pentru cel mai bun model CNN, am 4 straturi convoluționale cu 256, 64, 128 și 64 de filtre. Fiecare este urmat de un strat de poolin și un batch normalization.

După, am un strat de flattening și apoi un strat de dens cu 64 de unități și apoi un start dens pentru clasificarea binară. Pentru fiecare start funcția de activare este “relu”, în afară de ultimul care este “sigmoid”.

Detaliere parametrii

Conv2D(X,(3,3)) – are rolul de a extrage X caracteristici din poza de input

MaxPooling2D(2,2) – reduce dimensiunea hărții de caracteristici

BatchNormalization – normalizeaza datele de intrare din fiecare strat

Dropout(0.2) – pentru a evita overfitting, adaugă random noise

Flatten() – transformă inputul obținut într-un array uni-dimensional

Dense(64, activation = 'relu') – aplică o tranformare liniară datelor de input, dezactivând anumiți neuroni

Dense(1, activation = 'sigmoid') – produce clasificarea binara datorită funcției sigmoid

La final, am comparat rezultatele din y_pred cu 0.45 si 0.5, cele care depășesc numerele devenind 1. Pentru primele doua modele am obtinut un scor mai mare când am comparat cu 0.45 (datele tind să favorizeze 0), iar pentru ultimul un scor mai mare când am comparat cu 0.5.

De asemenea, am constatat că inițial că, mărind numărul de epoci, obțin o acuratețe mai mare. Dar, dacă sunt prea multe epoci, risc să fac overfitting (și să irosesc timp și resurse).

4. Concluzie

Modele aplicate sunt: Naïve Bayes, Random Forest și CNN. Naïve Bayes a avut o implementare facilă, dar nu rezultate excepționale, Random Forest a avut rezultate mai bune, dar e mai costisitor din punct de vedere al timpului, iar CNN a avut cele mai bune scoruri, dar pot duce către overfitting sau sub-învățare.

Nume model	*Hiperparametrii*	*Scor*
Naïve Bayes	Gaussian	*0.34*
Random Forest	*50 de copaci*	*0.44*
CNN First Try	*4 straturi, 5 epoci*	*0.32*
CNN – Second Try	*22 straturi, 50 epoci*	*0.49*
CNN – Third Try	*15 starturi, 15 epoci*	*0.51*