Documentație proiect – Inteligență Artificială

Furdui Vlad Rareș - Grupa 241

1. Scopul proiectului și înțelegerea datelor

Scopul acestui proiect este antrenarea unui model în vederea clasificării unor imagini ce reprezinta Computer Tomografe, astfel încât să se poată prezice cât mai exact dacă un creier prezintă sau nu o anomalie.

În următorul tabel se regăsesc numărul fiecărui set de date și ce reprezintă acesta:

Nume	Imagini_ct_train	Imagini_ct_valid	Imagini_ct_test	Indici_train	Indici_test
Mărime	15000	2000	5150	15000	2000
Descriere	Imagini pe fondul cărora se antrenează modelul	Imagini pe fondul cărora se probează modelul	Imagini pe fondul cărora se testează modelul	Label-uri care corespund cu imaginile pentru antrenament	Label-uri care corespund cu imaginile pentru validare

2. Metode de lucru folosite

În vederea obținerii unei corectitudini cât mai ridicată a predicției modelului, am folosit următorii doi clasificatori: **Multinomial Naive Bayes** și **Rețele Neuronale Convoluționare**

3. Multinomial Naive Bayes

Un algoritm ce se bazează pe concepte probalistice și care estimează, în urma clasificării elementelor anterioare probabilitatea unui element de a se încadra într-o clasă anume, acesta este patentat urmărind o eficiență crescută pentru seturi de cuvinte, însă am reușit să obțin o acuratețe bună cu acesta și pentru imagini. Pentru acest model, am obținut o acuratețe de 0.41

3.1 Biblioteci folosite

DENUMIRE	ROL
import numpy as np	Aceasta este folosită pentru transformarea listelor in np.array
from sklearn.metrics import f1_score	Este folosită pentru utilizarea funcției f1_score, metrică ce evaluează acuratețea modelului
from PIL import Image	Aceasta este folosită pentru prelucrarea modelului
from sklearn.naive_bayes import MultinomialNB	Prin intermediul acesteia se antrenează modelul Naive Bayes

3.2 Citirea datelor

Pentru o citire cât mai ușoară și clară a datelor am optat pentru o funcție, denumită "citire_de_date", care primește doi parametri care reprezintă intervalul de valori al imaginilor care vor fi citite în variabila set, ce ulterior returnează o listă. După ce calea este stabilită, imaginea este citită prin intermediul funcției „Image.open()", convertită în grayscale și memorează lista obținută pe post de numpy array în variabila set, transformând-o din 2D în 1D prin "flatten". Apoi, returnează variabila set.

```
def citire_de_date(st, sf):
    i = st
    set = []
    while i < sf :
        if i<10 :
            cale = "/kaggle/input/unibuc-brain-ad/data/data/00000"+str(i)+".png"
        elif i<100:
            cale = "/kaggle/input/unibuc-brain-ad/data/data/0000" + str(i) + ".png"
        elif i<1000:
            cale = "/kaggle/input/unibuc-brain-ad/data/data/000" + str(i) + ".png"
        elif i<10000:
            cale = "/kaggle/input/unibuc-brain-ad/data/data/00" + str(i) + ".png"
        else:
            cale = "/kaggle/input/unibuc-brain-ad/data/data/0" + str(i) + ".png"
        poza = Image.open(cale).convert('L')
        linie = np .array(poza)
        linie_np = np.array(linie).flatten()
        set.append(linie)
        i = i+1
    return set
```

Ulterior, este apelată funcția pentru imagini_ct train, imagini_ct_valid și imagini_ct test, fiecare având parametrii în legătură directă cu numărul de imagini designate pentru fiecare.

Am citit label-urile pentru train și valid fără a utiliza o funcție. Cele pentru train sunt citite prin intermediul list-comprehension, în timp ce, cele pentru valid doar sunt alăturate într-o listă ce inițial este nulă.

```
imagini_ct_train = citire_de_date(1, 17001)

imagini_ct_valid = citire_de_date(15001, 17001)


imagini_ct_test = citire_de_date(17001, 22150)
#apelarea functiei pentru cele trei seturi de date. Am citit si datele de valid in

indici_train = []
with open('/kaggle/input/unibuc-brain-ad/data/train_labels.txt', 'r') as f:
    f.readline()
    indici_train = [int(line.strip().split(",")[1]) for line in f][:15000]

indici_valid = []
with open('/kaggle/input/unibuc-brain-ad/data/validation_labels.txt', 'r') as f:
    f.readline()
    for line in f.readlines():
        indici_valid.append(int(line.strip().split(',')[1]))
        indici_train.append(int(line.strip().split(',')[1]))
```

3.3 Împărțirea pe intervale

Ulterior, în vederea utilizării algoritmului Naive-Bayes, am împărțit în câte **4 intervale** valorile de la 0 la 230. Inițial, numărul de intervale era mult mai ridicat, în jurul a 1947, însă cele mai bune rezultate le-am obținut cu doar 4. Totodată, în ciuda faptului că se poate ridica până la 255, am ales 230 din același raționament bazat pe eficiență.

Transformarea în intervale are loc prin intermediul funcției *valori_spre_intervale*, care primește o listă cu listele fiecărei categorii de imagini transformate în numpy array, respectiv variabila intervale și returnează intervalele în care aceasta se încadrează. Variabila intervale este inițiată cu împărțirea menționată anterior prin intermediul funcției linspace.

```
intervale  = np.linspace(0, 230, num = 4)
imagini_ct_valid = np.array((imagini_ct_valid))
imagini_ct_train = np.array((imagini_ct_train))
imagini_ct_test = np.array((imagini_ct_test))
#transformarea in np.array pentru a putea lucra cu ele
interval_train = valori_inspre_intervale(imagini_ct_train,intervale)
interval_train = interval_train.reshape(interval_train.shape[0], -1)

interval_valid = valori_inspre_intervale(imagini_ct_valid,intervale)
interval_valid = interval_valid.reshape(interval_valid.shape[0], -1)

interval_test = valori_inspre_intervale(imagini_ct_test, intervale)

interval_test = interval_test.reshape(interval_test.shape[0], -1)
```

Valori_spre_intervale creează inițial un array doar cu valori nule, ca mai apoi să atribuie intervalul corespunzător prin intermediul funcției np.digitize().

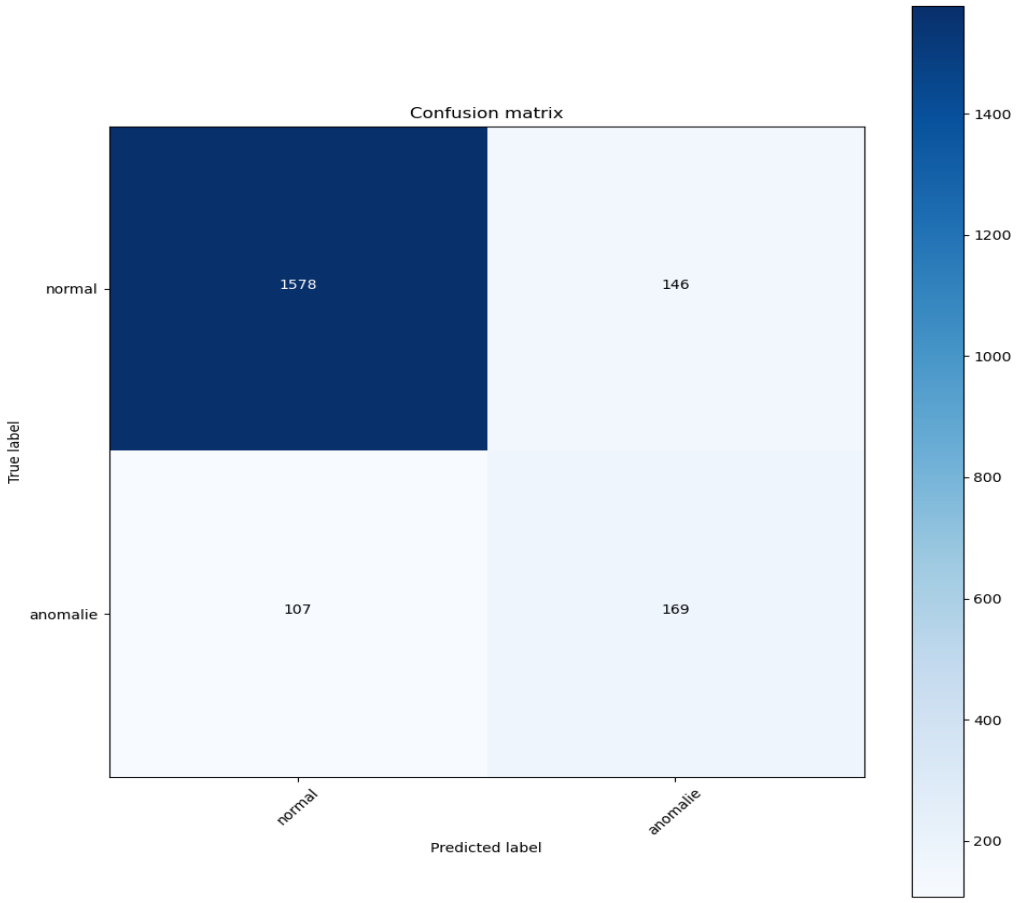
```
def valori_inspre_intervale(poze, intervale):
        poze_noi = np.zeros(poze.shape)
        for i in range(poze.shape[0]):
            poze_noi[i] = np.digitize(poze[i], intervale)
        return poze_noi - 1
```

3.4 Antrenare și afișare

Ulterior, naive_bayes_model va reprezenta modelul creat cu funcția MultinomialNB, la care am adăugat un hiperparametru cu valoarea 3.2, antrenat cu imaginile și label-urile de train, apoi verificat cu cele de valid. Ulterior, se va efectua o predicție pentru test, pe care o vom exporta într-un csv prin intermediul funcției din numpy savetxt.

```
naive_bayes_model = MultinomialNB(alpha = 3.2)
#apelarea functiei pentru Naive Bayes in care am introdus si un hiperparametru de 3.2
naive_bayes_model.fit(interval_train, indici_train)
#antrenarea modelului
print(naive_bayes_model.score(interval_valid, indici_valid))
indici_predictie = naive_bayes_model.predict(interval_test)

#apelarea functiei de predict pentru a obtine numarul de predictii
lista_afisare=[["id", "class"]]
for i in range(17001, 22150):
    lista_afisare.append([int(i), int(indici_predictie[i-17001])])
#introducerea in lista_afisare datele pentru csv si ulterior salvarea fisierului
np.savetxt("Competitie.csv", lista_afisare, fmt="%s", delimiter= ',')
```

Aceasta este matricea de confuzie a modelului :



4. Convolutional Neural Network

Folosind mai multe straturi convoluționare pentru a observa șabloane spațiale, acest algoritm oferă posibilitatea unei eficiențe ridicate pentru clasificarea imaginilor. Cu acesta am obținut o acuratețe de 0.37.

4.1 Biblioteci folosite

Nume	Rol
import numpy as np	Aceasta este folosită pentru transformarea listelor in np.array
from PIL import Image	Aceasta este folosită pentru prelucrarea modelului
import tensorflow as tf	Folosit pentru antrenare și validare
from tensorflow import keras	Folosit pentru inițierea rețelelor neuronale
from tensorflow.keras.models import Sequential	Folosit pentru inițierea liniară a straturilor
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout	Diferite clase care ajută la formarea rețelelor neuronale

4.1 Citirea datelor

Din punct de vedere al citirii datelor, aceasta este identică în comparație cu celălalt model.

4.2 Crearea modelului

Din punct de vedere al straturilor folosite, am utilizat:

- Un **strat convoluționar** cu 32 filtre, de dimensiune 3x3, activat de către funcția ReLU. Întrucât este primul strat, trebuie specificat un input shape de (224,224,1)
-Acesta este urmat de un **MaxPooling2D()** care are rolul de diminuare a datelor, căutând maximul fiecărui kernel de 2x2

Acest patten se repetă de încă 2 ori, diferența făcând-o numărul de filtre din stratul convoluționar, 128 respectiv 256

-Continuând, este adăugat un strat **Flatten**, ce leagă straturile conectate total de ceea ce rezultă

din straturile inițiale, transformând rezultatul într-un vector 1D

- Stratul Flatten este urmat de 3 straturi **Dense**, conectate total la output, cu 256, 128 si 32 de unități, urmat de un strat de tip Dropout, care are rol de prevenire a overfitting. Având rata de dropout 0.25, acesta va seta aleator 25% din input units cu valoarea 0.

-Ultimul strat este de asemenea de tip Dense, acitvat cu *Sigmoid*, ce contribuie la deciderea apartenenței la o clasă a imaginii curente.

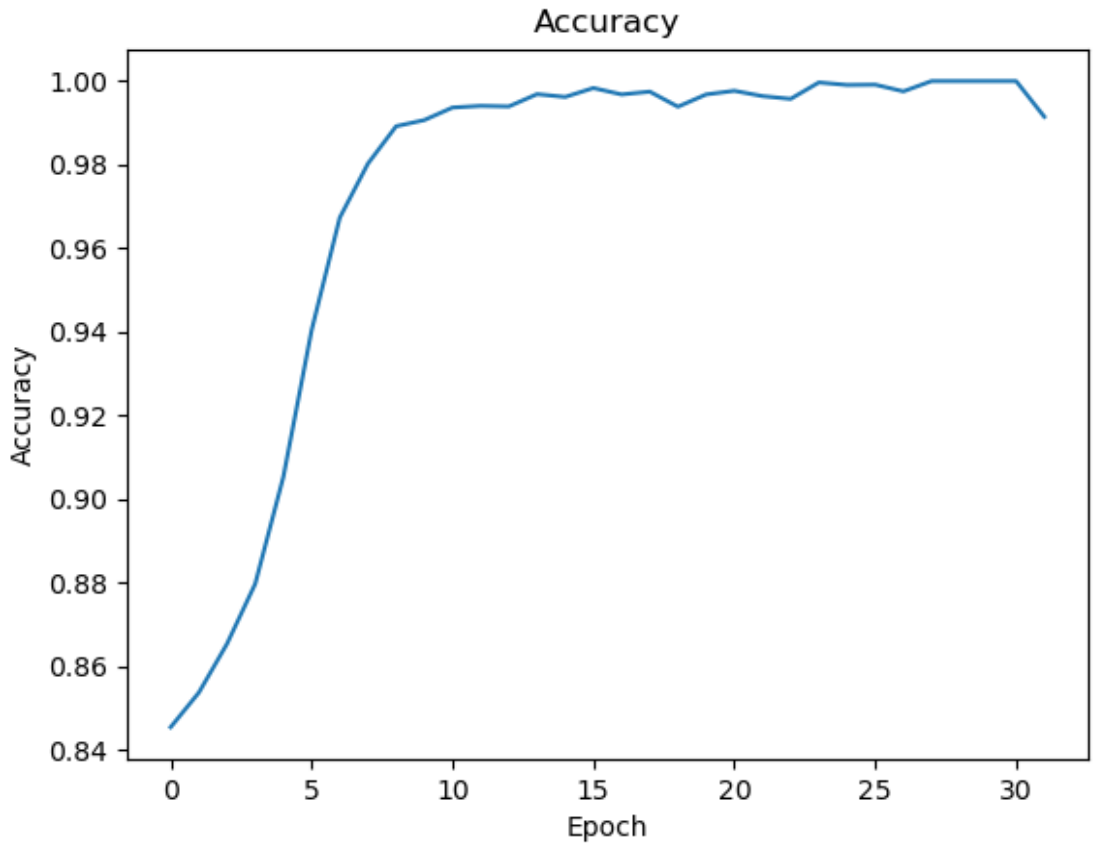
```
model= Sequential([


Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 1)),
MaxPooling2D(),


Conv2D(128, (3, 3), activation='relu'),
MaxPooling2D(),

Conv2D(256, (3, 3), activation='relu'),
MaxPooling2D(),

Flatten(),
Dense(256, activation='relu'),
Dense(128, activation='relu'),
Dense(32, activation='relu'),
Dropout(0.25),
Dense(1, activation='sigmoid')])
```

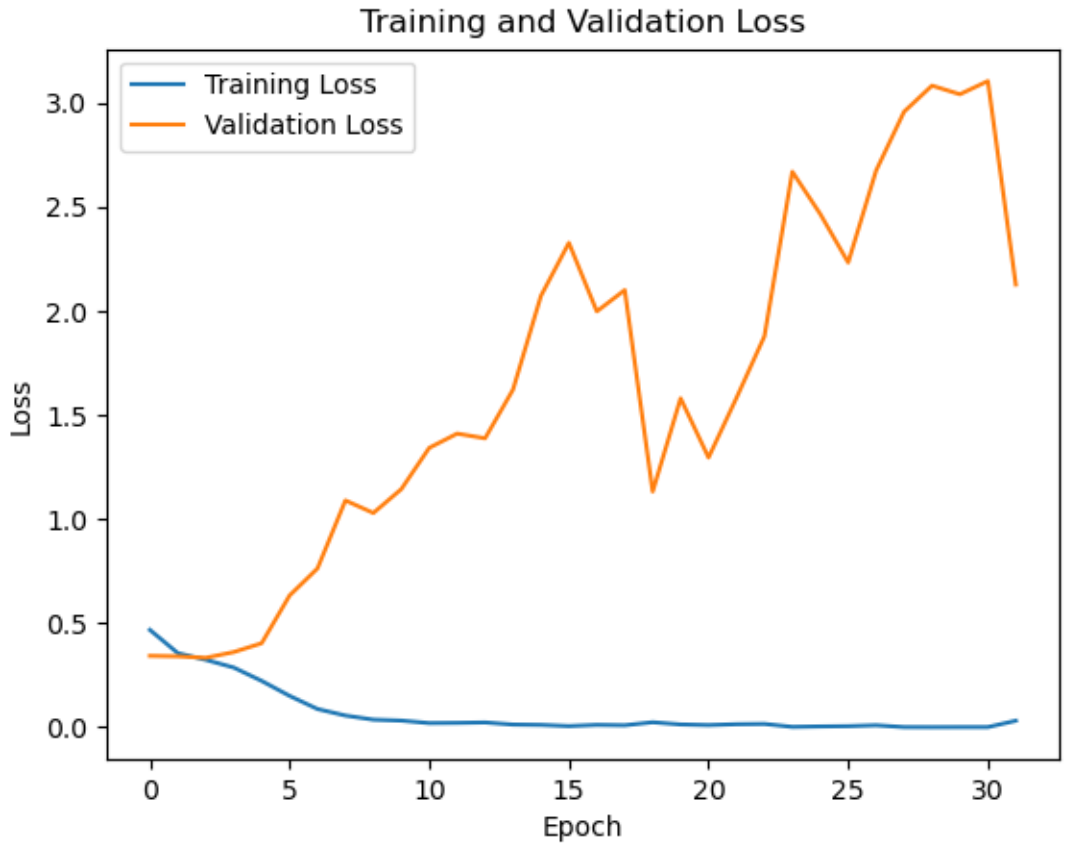
4.3 Compilare, antrenare și prezicere

Ulterior, modelul este compilat cu optimizatorul *Adam,* cunoscut ca fiind unul dintre cele mai bune pentru clasificarea imaginilor, BinaryCrossentropy pe post de funcție de pierde și ca metrică de acuratețe am optat pentru accuracy, pentru determinarea proporțională a imaginilor clasificate bine.

Apoi vom antrena modelul cu imaginile și valorile de train, cu un număr de 32 de epoci și datele împărțite în 32 de batch-uri, validând acuratețea cu imaginile și label-urile pentru valid.

Am ales aceste valori în urma acestor 2 grafice:

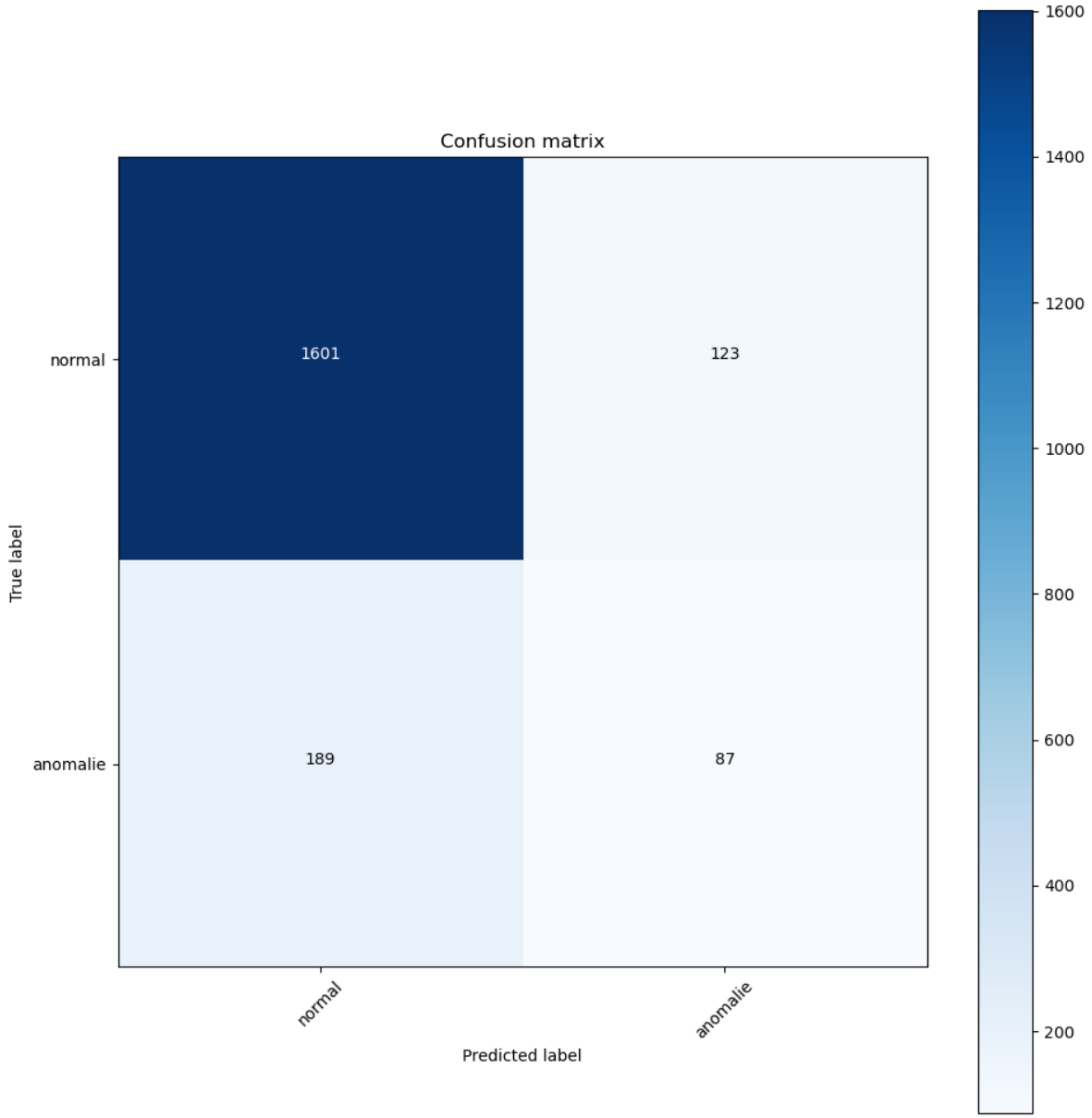




Ulterior, modelul va face o predicție în baza datelor din imagini_ct_test, dând valori ce trebuie clasificate ulterior ca fiind 0 sau 1.

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history = model.fit(np.array(imagini_ct_train).reshape(-1, 224, 224, 1),
                    np.array(indici_train),
                    epochs=32,
                    batch_size=32,
                    validation_data=(np.array(imagini_ct_valid).reshape(-1, 224, 224, 1),
                    np.array(indici_valid)))
indici_predictie = model.predict(np.array(imagini_ct_test).reshape(-1, 224, 224, 1))
print(indici_predictie)
for i in range (len(indici_predictie)):
                if indici_predictie[i] < 0.5:
                        indici_predictie[i] = 0
                else:
                    indici_predictie[i] = 1
                print(indici_predictie[i])


lista_afisare=[["id", "class"]]
for i in range(17001, 22150):
    lista_afisare.append([int(i), int(indici_predictie[i-17001])])
np.savetxt("Competitie.csv", np.array(lista_afisare), fmt="%s", delimiter= ',')
```

În final, rezultatul este exportat într-un csv.
Aceasta este matricea de confuzie a modelului:



5. Concluzii

În concluzie, în urma încercării a celor două metode prezentate, Naive Bayes și Convolutional Neural Network, am reușit să obțin cea mai bună acuratețe ca fiind una de 0.41.