## Prezentarea problemei

Cerința proiectului presupune antrenarea unui clasificator de imagini ce ilustrează tomografii ale creierului uman, imagini de dimensiuni 224 x 224 pixeli, ce se împart în 2 categorii: anomalie (label 1) și normal (label 0).

Datele puse la dispoziție conțin 15000 de imagini de antrenament, 2000 de imagini de validare, respectiv label-urile acestora. Imaginile de testare sunt în număr de 5149.

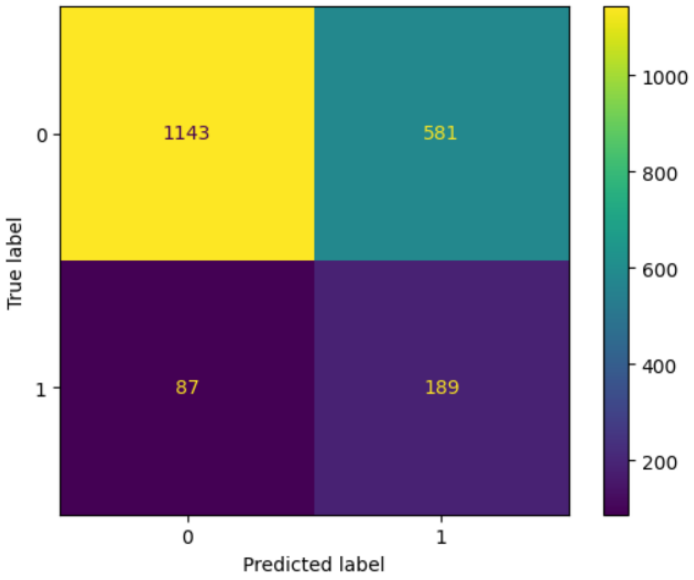
O caracteristică remarcată la setul de date este discrepanța dintre ponderea claselor: doar în jur de 15% din imaginile de antrenament aparțin clasei 1, în timp ce restul de aproximativ 85% aparțin clasei 0, ceea ce transformă problema într-o problemă cu set de date imbalansat. Datele de validare sunt împărțite astfel: 13.8% au label-ul 1, restul de 86,2% au label-ul 0.

## Abordările propuse

- Naive Bayes, scor final 0.44536  (0.41558 inițial)
- KNN, scor final 0.22366 (0.13114 inițial)
- OneClassSVM, scor final 0.22323 (0.20763 inițial)
- CNN, scor final 0.68767 (0.70038 inițial)

## Naive Bayes

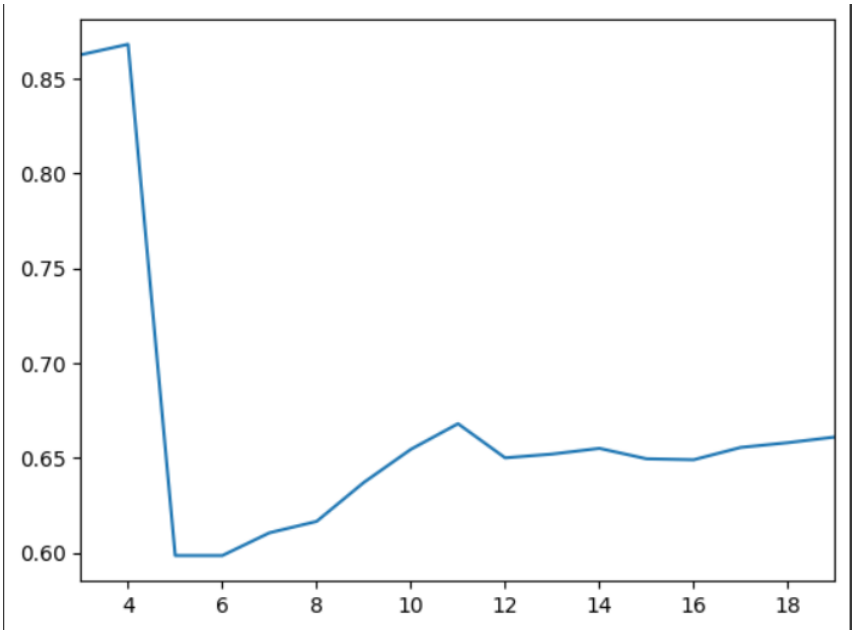
Prima abordare a fost modelul Naive Bayes, după structura prezentată la laborator. Scorul maxim obținut cu acest model este de 0.41558 pe platformă. Prima submisie realizată folosind acest algoritm nu a inclus metoda values_to_bins folosită la laborator și adăugată ulterior. Pentru normalizarea imaginilor am folosit inițial împărțirea pixelilor la 255, fără nicio altă preprocesare, după care am antrenat clasificatorul MultinomialNB. Pentru citirea imaginilor din fișier am folosit librăria PIL alături de metoda convert(„L”), care le convertește la formatul alb-negru. Folosind această abordare, pe datele de validare am obținut un scor de 0.666, cu următoarea matrice de confuzie:



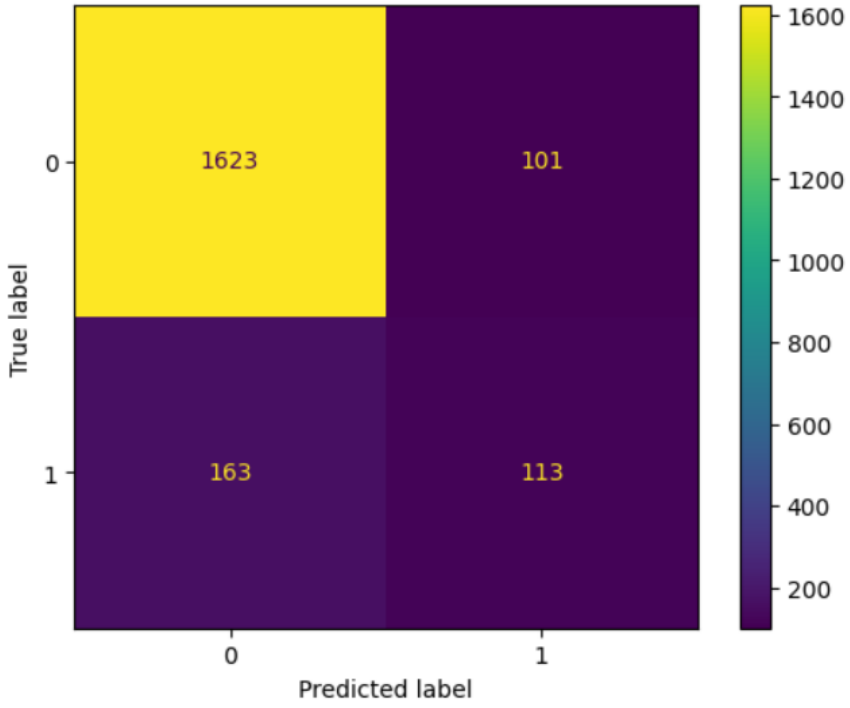

Această primă încercare a rezultat într-un scor de 0.33395 pe platformă, o precizie de 0.2454, senzitivitate de 0.6847 și specificitate de 0.6629, după cum urmează:

| | Precision | Recall |
|---|---|---|
| 0 | 0.9292 | 0.6629 |
| 1 | 0.2454 | 0.6847 |

Următoarea încercare cu același algoritm s-a folosit de metoda values_to_bins și metoda digitize pentru a împărți valorile continue în intervale. Pentru a găsi numărul de bins optim, am evaluat scorul obținut pe datele de validare în funcție de numărul de bins, de la 3 până la 19 inclusiv. Graficul rezultat arată astfel:



Maximul este obținut cu numărul de bins = 4, scorul fiind de 0.868, după care acuratețea scade spre 0.64-0.66. Matricea de confuzie arată astfel:




Precision score a fost de aprox. 0.528, senzitivitatea/recall-ul (abilitatea de a prezice clasa 1) de 0.4094, iar specificitatea (abilitatea de a prezice clasa 0) de 0.9414.

| Class | Precision | Recall |
|---|---|---|
| 0 | 0.9087 | 0.9414 |
| 1 | 0.5280 | 0.4094 |

Această metodă a obținut pe platformă scorul de 0.41558.

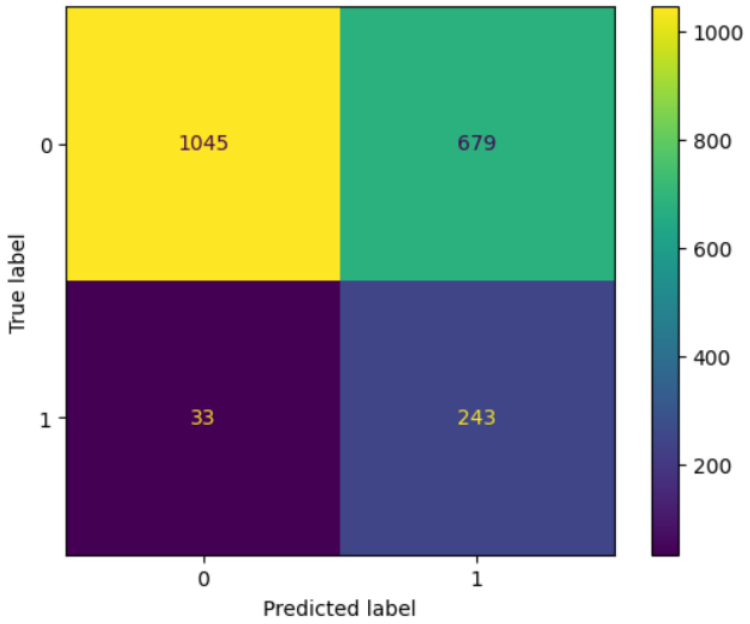
## KNN

Următorul algoritm încercat a fost KNN, implementat asemenea cerințelor prezentate la laborator. Procesarea imaginilor a rămas neschimbată de la metoda Naive Bayes. Din cauza timpului de rulare foarte ridicat (o singură iterație prin imaginile de test și generare de predicții a durat aproximativ 5h, în ciuda

încercărilor de optimizare), unica submisie realizată a folosit parametrii L2 și un număr de 3 vecini, având pe platformă scorul de 0.13114.

## OneClassSVM

În urma research-ului efectuat, a reieșit că o posibilă rezolvare a problemei poate fi metoda de OneClassSVM care se specializează în detectarea anomaliilor, pornind de la premisa că datele normale ar trebui să se găsească într-o regiune densă a spațiului de date, iar punctele anormale (anomaliile) ar trebui să fie departe de această regiune. Din imaginile de antrenament le-am extras pe cele care aparțin clasei 0, și am antrenat modelul folosind acest set de date. Folosind metoda score_samples, am obținut scorul de anomalie de 1141.57 pe imaginile de test, acest număr reprezentând măsura în care imaginile de test se abat de la caracteristicile clasei 0. Din imaginile de test, 2579 (mai mult de jumătate) au fost clasificate ca aparținând clasei 1. Matricea de confuzie arată astfel:




Pe acest model am obținut pe platformă scorul de 0.20763.

| Class | Precision | Recall |
|---|---|---|
| 0 | 0.6061 | 0.969 |
| 1 | 0.8804 | 0.3578 |

## CNN versiunea 1

Ultimul model încercat a fost Convolutional Neural Network, caracterizat prin existența a 3 layere specifice lui: layerul convoluțional, layer de pooling și layerul dens, unde are loc clasificarea propriu-zisă.

Unul dintre primele modele care mi-a îmbunătățit scorul de pe platformă (obținând 0.52542) a avut următoarea structură de layere:

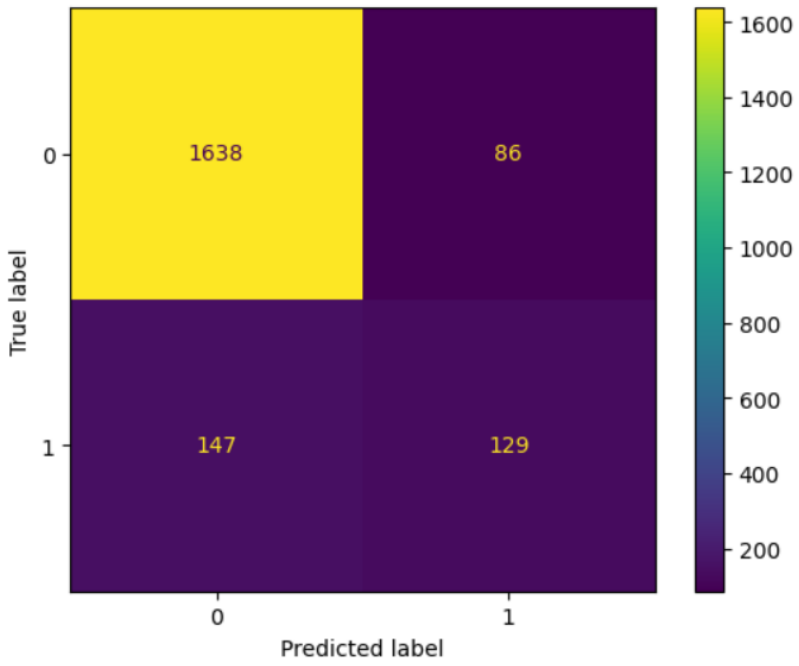
| | |
|---|---|
| Conv2D | filters=16, kernel_size=(3, 3), activation='relu', input_shape=(224, 224, 1) |
| MaxPooling2D | pool_size=(2, 2) |
| Conv2D | filters=32, kernel_size=(3, 3), activation='relu' |
| MaxPooling2D | pool_size=(2, 2) |
| Conv2D | filters=64, kernel_size=(3, 3), activation='relu' |
| MaxPooling2D | pool_size=(2, 2) |
| Conv2D | filters=64, kernel_size=(3, 3), activation='relu' |
| MaxPooling2D | pool_size=(2, 2) |
| Dropout | 0.1 |
| Flatten | default |
| Dense | 64, activation='relu' |
| Dense | 64, activation='relu' |
| Dense | 64, activation='relu' |
| Dense | 64, activation='relu' |

| | |
|---|---|
| Dense | 2, activation='softmax' |

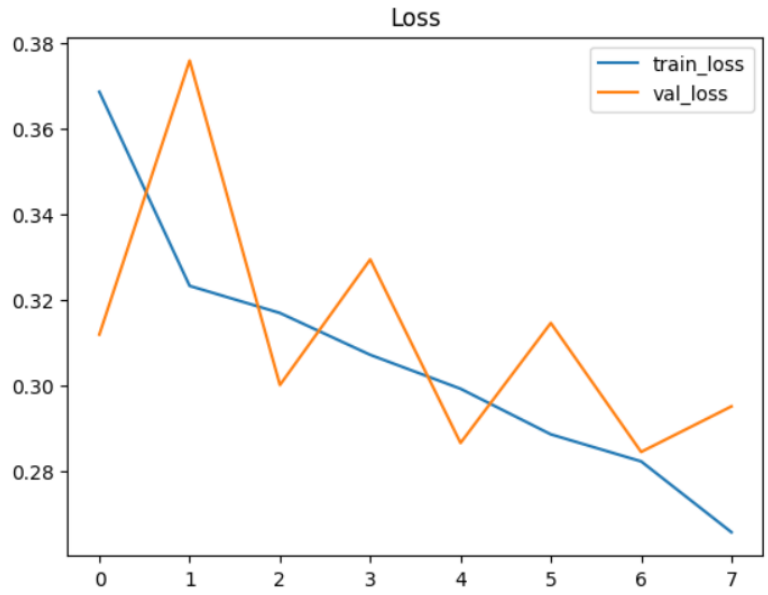
La început, pentru normalizarea imaginilor, am rămas pe metoda împărțirii pixelilor la 255. Am antrenat modelul timp de 8 epoci, ceea ce la momentul respectiv a dus la un loss de 0.295 și o acuratețe de 0.883 pe datele de validare, cu precision și recall-ul următoare:

| Class | Precision | Recall |
|---|---|---|
| 0 | 0.9176 | 0.9501 |
| 1 | 0.6s | 0.467 |

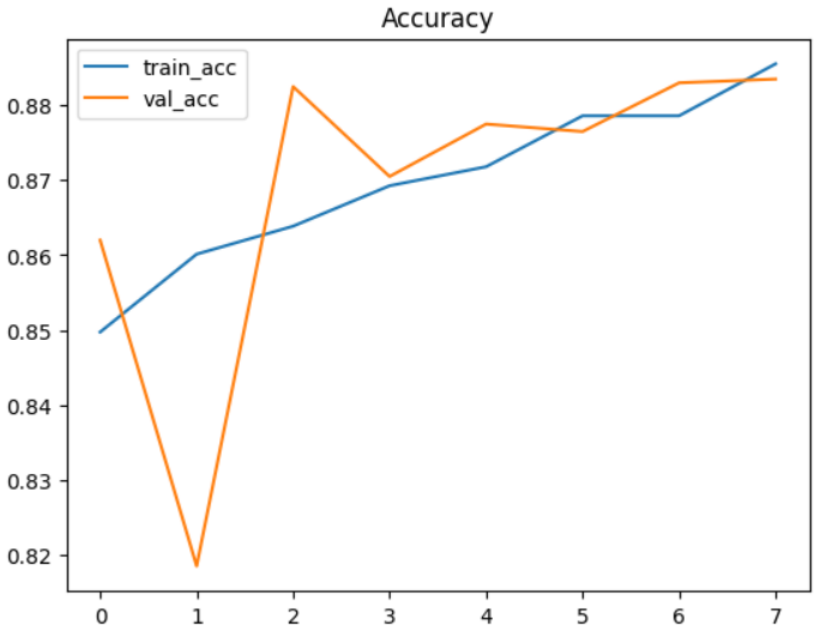
Matricea de confuzie:



Graficul pentru loss a arătat în felul următor:




Graficul pentru acuratețe:




## CNN versiunea 2

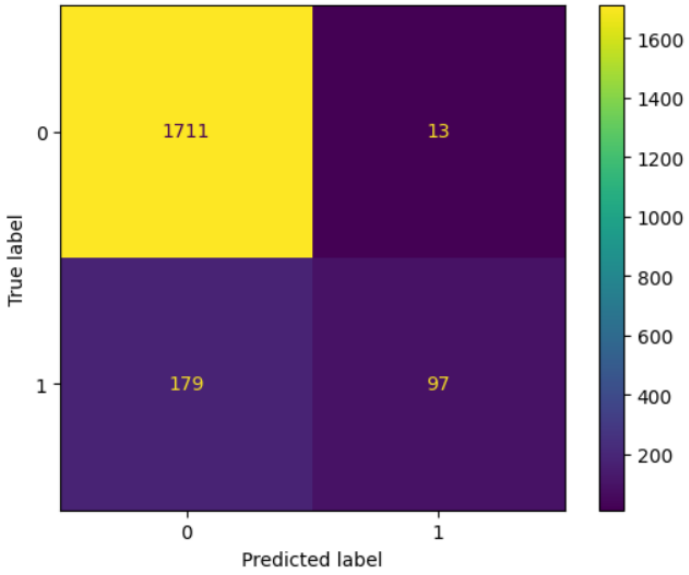
În următorul model de CNN încercat, care a crescut la scorul de 0.5862 pe Kaggle, am încercat câteva metode noi. În primul rând, după analiza imaginilor, am observat că toate tomografiile sunt înconjurate de o margine neagră care nu contribuie cu nimic la clasificare, motiv pentru care am micșorat dimensiunile acestora la 176 x 160 de pixeli, în încercarea de a nu lăsa modelul să extragă astfel

de caracteristici irelevante. Pentru procesarea imaginilor am folosit librăria scikit learn – preprocessing – normalize, cu norma L2. M-am folosit de hiperparametrul de callbacks de la metoda fit a modelului și de ModelCheckpoint pentru a salva modelul care are cea mai mare acuratețe pe datele de validare. Am antrenat modelul timp de 30 de epoci. De asemenea, am sporit numărul de layere astfel:

| | |
|---|---|
| Conv2D | filters=64, kernel_size=(3,3), activation='relu' |
| BatchNormalization | |
| MaxPooling2D | pool_size=(2, 2) |
| Conv2D | filters=128, kernel_size=(3,3), activation='relu' |
| BatchNormalization | |
| MaxPooling2D | pool_size=(2, 2) |
| Dropout | 0.1 |
| Conv2D | filters=256, kernel_size=(3,3), activation='relu' |
| BatchNormalization | |
| MaxPooling2D | pool_size=(2, 2) |
| Dropout | 0.1 |
| Conv2D | filters=512, kernel_size=(3,3), activation='relu' |
| BatchNormalization | |
| MaxPooling2D | pool_size=(2, 2) |
| Flatten | |
| Dense | 512, activation='relu' |

| | |
|---|---|
| BatchNormalization | |
| Dropout | 0.2 |
| Dense | 256, activation='relu' |
| BatchNormalization | |
| Dropout | 0.2 |
| Dense | 128, activation='relu' |
| BatchNormalization | |
| Dropout | 0.2 |
| Dense | 64, activation='relu' |
| BatchNormalization | |
| Dropout | 0.2 |
| Dense | 2, activation='sigmoid' |

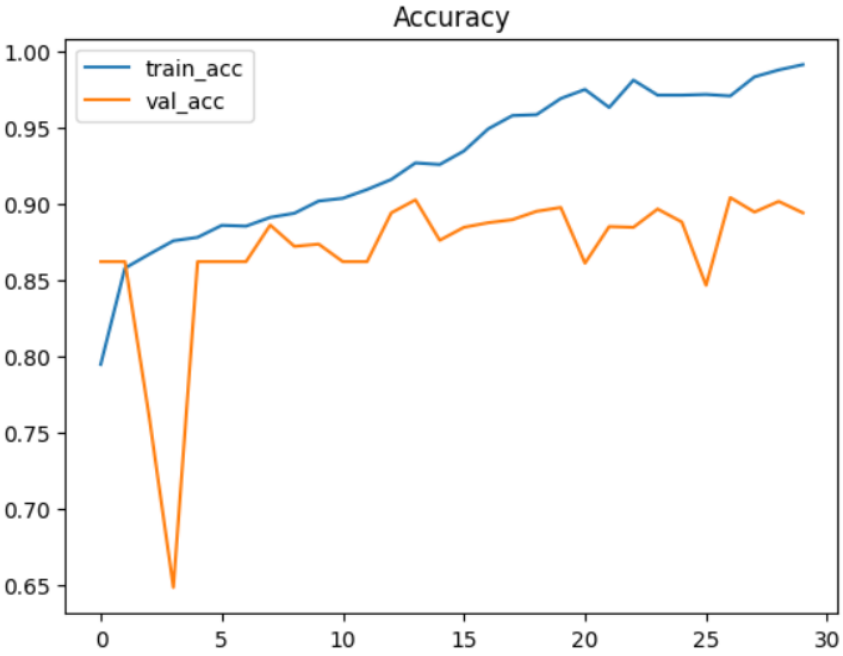
Modelul cel mai bun a fost salvat la epoca 27 și a avut o acuratețe de 0.904, cu următoarea matrice de confuzie (dreapta):



| Class | Precision | Recall |
|---|---|---|
| 0 | 0.9052 | 0.9924 |
| 1 | 0.8818 | 0.3514 |

Graficul loss-ului:


Graficul acurateței:



## CNN versiunea 3

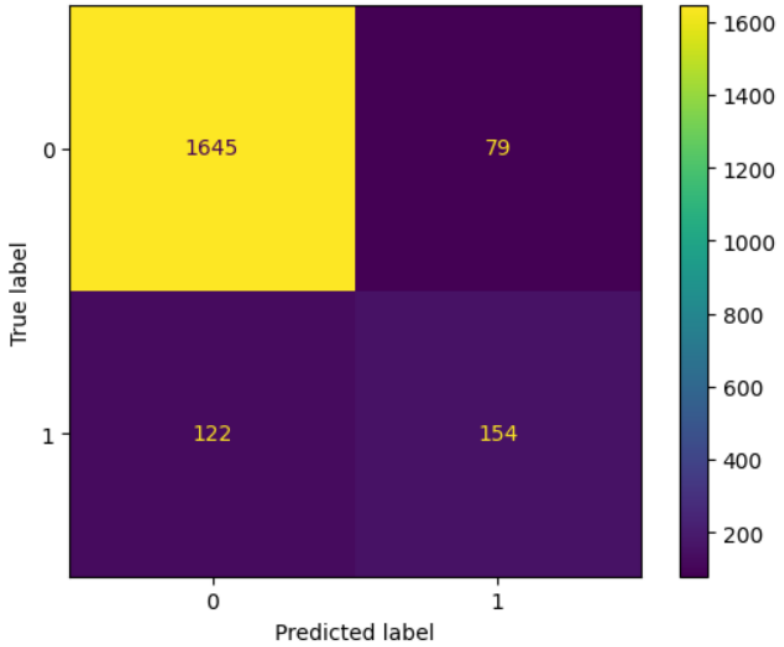
După mai multe submisii în care deși aveam acurateţe de peste 0.9 pe datele de validare, scorul pe Kaggle nu creștea, am realizat că acuratețea nu este cel mai de încredere indicator al performanței unui model de clasificare binară în care există dezechilibru între clase, deoarece chiar și dacă ar prezice că toate imaginile aparțin clasei majoritare și niciuna clasei minoritare, acuratețea ar fi încă destul de ridicată. Astfel am decis să îmi suprascriu metoda de salvare a celui mai bun model, pe baza observației că un model mai bun ar prezice undeva între 515 și 800 de label-uri de 1 în output, adică între 10% și 15.5% din imaginile de test. O altă condiție de salvare a modelului a fost ca scorul AUC să fie peste 0.899 (deoarece la model.compile am avut ca metrics AUC() și accuracy, în această ordine). Arhitectura modelului de CNN a rămas aceeași cu cea prezentată anterior, iar funcția personalizată care salva modelul este următoarea:

```
numar = 0

class Save_Best(tf.keras.callbacks.Callback):
    def test_model(self, epoch, logs={}):
        global numar
        global test_images
        #obtin predictiile pe imaginile de test
        predictions = self.model.predict(test_images)
        #evaluez performanta pe datele de validare
        evaluare = self.model.evaluate(validation_images, validation_labels)
        #numar cati de 1 am in predictiile pe test_images
        count = 0
        for prediction in predictions:
          if np.argmax(prediction) == 1:
            count += 1
        print(count)
        #salvez modelul doar daca indeplineste aceste conditii simultan si doar
        #daca are mai multi de 1 in predictii fata de modelul salvat anterior
        if count > numar and 515 < count and 800 > count and evaluare[1] >= 0.899:
            numar = count
            self.model.save('most_1s_model.h5')
            print(f'Salvat model cu {count} de 1 in output.')
```

Cea mai bună variantă a acestui model antrenat 30 de epoci a prezis 621 de apariții a clasei 1 în output, ceea ce a rezultat într-un scor de 0.60337 în competiție. A avut loss de 0.362, scor AUC de 0.951 și acuratețe de 0.8995.

Matricea de confuzie aferentă:




| Class | Precision | Recall |
|---|---|---|
| 0 | 0.9309 | 0.9541 |
| 1 | 0.6609 | 0.5579 |

## CNN versiunea 4

Pentru această versiune a rețelei neuronale am făcut doar mici schimbări în funcția de salvare a celui mai bun model. Am folosit matricea de confuzie calculată pentru datele de validare și am făcut suma pe diagonala principală pentru a vedea câte imagini au fost clasificate corect.

Condițiile noi de salvare a unui model, pe lângă condiția ca numărul de label-uri de 1 din output să fie între 515 și 800:

- Numărul de imagini clasificate corect la un anumit pas să fie mai mare decât numărul de imagini clasificate corect al modelului salvat anterior.
- Numărul de imagini cu clasa 1 clasificate corect să fie peste 140.
- Acurateţe peste 0.88.

```
numar = 0
total_corect = 0

class Save_Best(tf.keras.callbacks.Callback):
    def test_model(self, epoch, logs={}):
        global numar
        global test_images
        global total_corect
        predictions = self.model.predict(test_images)
        predictions_validation = self.model.predict(validation_images)
        evaluare = self.model.evaluate(validation_images, validation_labels)

        #calcul matrice de confuzie
        conf_matrix = confusion_matrix([np.argmax(pred) for pred in validation_labels], [np.argmax(pred) for pred in predictions_validation])
        print(conf_matrix)

        #calcul nr de 1 din output
        count = 0
        for prediction in predictions:
          if np.argmax(prediction) == 1:
            count += 1
        print(count)
        #conditiile de salvare model, evaluare[2] este acuratetea
        if conf_matrix[0][0] + conf_matrix[1][1] >= total_corect and 500 < count and 800 > count and evaluare[2] >= 0.88 and conf_matrix[1][1] >= 140:
            total_corect = conf_matrix[0][0] + conf_matrix[1][1]
            self.model.save('best_model.h5')
            print(f'Salvat model cu {count} de 1 in output.')
```

După antrenarea modelului timp de 50 de epoci, cea mai bună versiune a prezis 529 de etichete 1 în output, cu un loss de 0.584, scor AUC de 0.9437 și acuratețe 0.896. Matricea de confuzie, și prin urmare și recall și precision au fost, în mod surprinzător, egale cu valorile obținute de modelul CNN versiunea 3.

## CNN versiunea 5

Pentru a crea un model mai performant am încercat diverse tehnici de augmentare a datelor. Inițial, am încercat ImageDataGenerator cu diverși hiperparametri: flip, rotation_range, zoom_range, etc, însă performanța nu s-a îmbunătățit. Ulterior, am încercat echilibrarea setului de date prin crearea artificială de imagini ce aparțin clasei 1. Am utilizat zgomote speckle și gaussian, aplicate imaginilor ce aparțin clasei 1. Extragerea acestor imagini arăta în felul următor:

```
def extract_label_1_images(images, labels):
  final_images = []
  final_labels = []
  for i, label in enumerate(labels):
    if label[1] == 1:
      #adaug imaginile care au clasa 1
      final_labels.append([0,1])
      final_images.append(images[i])
  return np.array(final_images), np.array(final_labels)

train_images_1, train_labels_1 = extract_label_1_images(train_images, train_labels)
```

Funcția de adăugare zgomot speckle:

```
def add_speckle():
  global train_images_1
  global train_images
  global train_labels

  #setez o medie
  speckle_mean = 0
  #setez deviatia standard
  speckle_std = 0.2

  #setez forma speckle-ului
  speckle_shape = (len(train_images_1), 176, 160, 1)
  #generez valori aleatoare ce urmeaza distributia normala
  speckle = np.random.normal(speckle_mean, speckle_std, speckle_shape)

  #aplicare zgomot
  noisy_images = train_images_1 + train_images_1 * speckle
  #limitez rezultatul intre 0 si 255
  noisy_images = np.clip(noisy_images, 0, 255).astype(np.uint8)
  max_value = np.max(noisy_images)
  #normalizare imagini prin impartire la pixelul cu cea mai mare valoare
  noisy_images = noisy_images / max_value
  # concatenare cu imagini originale
  train_images = np.concatenate([train_images, noisy_images], axis=0)
  train_labels = np.concatenate([train_labels, [[0,1] for i in range(len(train_images_1))]], axis=0)


add_speckle()
```

Funcția de adăugare zgomot gaussian:

```
def add_gaussian():
  global train_images_1
  global train_images
  global train_labels

  #setez media
  mean = 0
  #setez deviatia
  stddev = 25

  #generare valoare aleatoare
  noise = np.random.normal(mean, stddev, (176,160,1))
  noisy_images = train_images_1 + train_images_1 * noise

  #limitez rezultatul intre 0 si 255
  noisy_images = np.clip(noisy_images, 0, 255).astype(np.uint8)
  max_value = np.max(noisy_images)
  #normalizare
  noisy_images = noisy_images / max_value

  #concatenare cu imaginile si label-urile initiale
  train_images = np.concatenate([train_images, noisy_images], axis=0)
  train_labels = np.concatenate([train_labels, [[0,1] for i in range(len(train_images_1))]], axis=0)

add_gaussian()
```

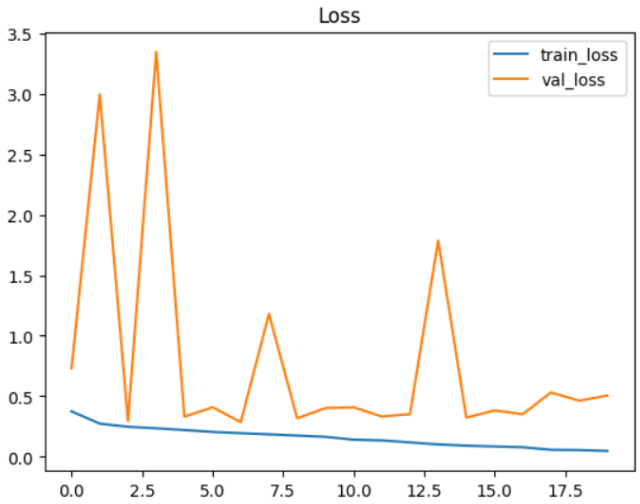
Metoda de salvare a celui mai bun model a rămas neschimbată de la modelul anterior. Deși am antrenat modelul timp de 40 de epoci, nu a existat nicio instanță

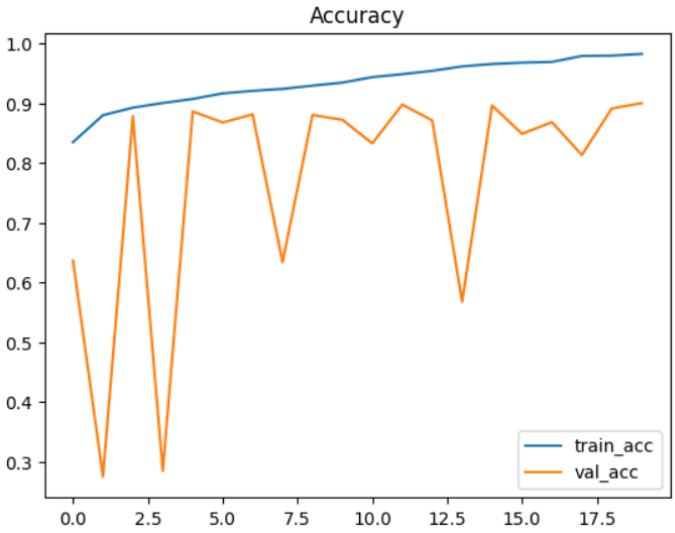
a acestuia care să satisfacă cerințele de salvare, motiv pentru care nu am mai continuat pe această metodă.

O altă metodă de augmentare a datelor a fost folosirea ADASYN, care generează noi exemple din clasele minoritare. Am setat sampling_strategy = minority pentru a specifica că doresc doar exemple sintetice pentru clasa minoritară. Secvența de cod care face acest lucru:

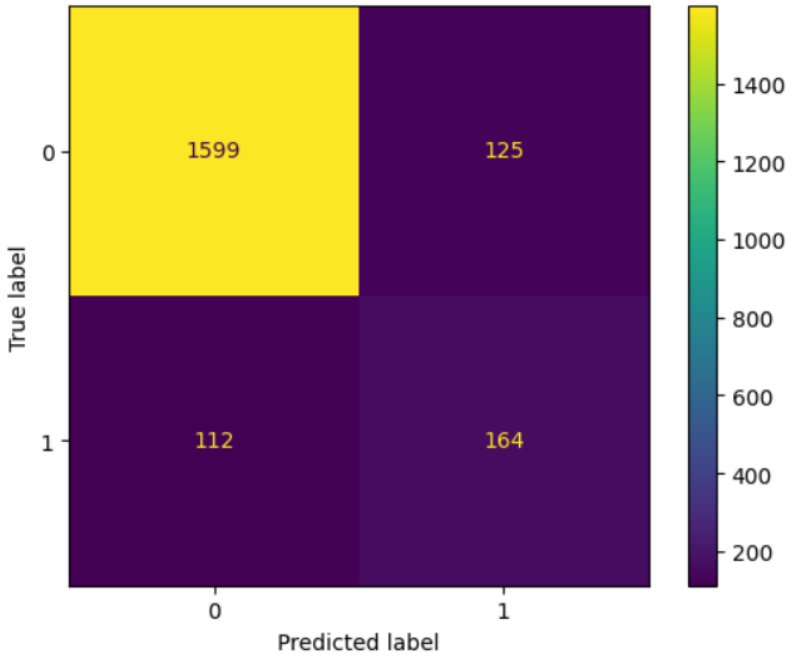
```
#ADASYN
adasyn = ADASYN(sampling_strategy='minority')
train_images, train_labels = adasyn.fit_resample(train_images.reshape(-1, 176 * 160), train_labels)
```

Am antrenat modelul 20 de epoci, iar modelul salvat a avut 738 de label-uri de 1 în output care a dat un scor de 0.59166 pe Kaggle, ceea ce nu a fost o îmbunătățire. Graficele pierderii și acurateței:



Matricea de confuzie:




| Class | Precision | Recall |
|---|---|---|
| 0 | 0.9345 | 0.9274 |
| 1 | 0.5674 | 0.5942 |

Deoarece am observat fluctuații mari pe acuratețea datelor de validare în timpul procesului de antrenare, am decis să nu mai utilizez Adasyn în continuare.

## CNN versiunea finală

Deoarece cu un singur model nu am reușit să depășesc pragul la care m-am plafonat (0.62), am decis să combin mai multe modele astfel: antrenez un model pe 50 de epoci și salvez cele mai bune 5 instanțe ale sale. Inițial, păstrând condițiile de salvare folosite anterior, de cele mai multe ori nici nu ajungeau să fie salvate 5 modele care să întrunească toate cerințele, așa că am eliminat treptat din ele. Era suficient ca un model să prezică cu acuratețe câteva poziții de 1, deoarece s-ar fi completat cu predicțiile celorlalte modele. Din cele 5 modele rezultate, la final le combinam predicțiile făcând suma tuturor celor 5 predicții de pe o anumită poziție i. Astfel că, de exemplu, dacă suma pe o poziție era 3, înseamnă că 3 modele au prezis ca imaginea de pe acea poziție aparține clasei 1. Abordarea inițială a fost de a lua predicțiile în funcție de votul majoritar (suma

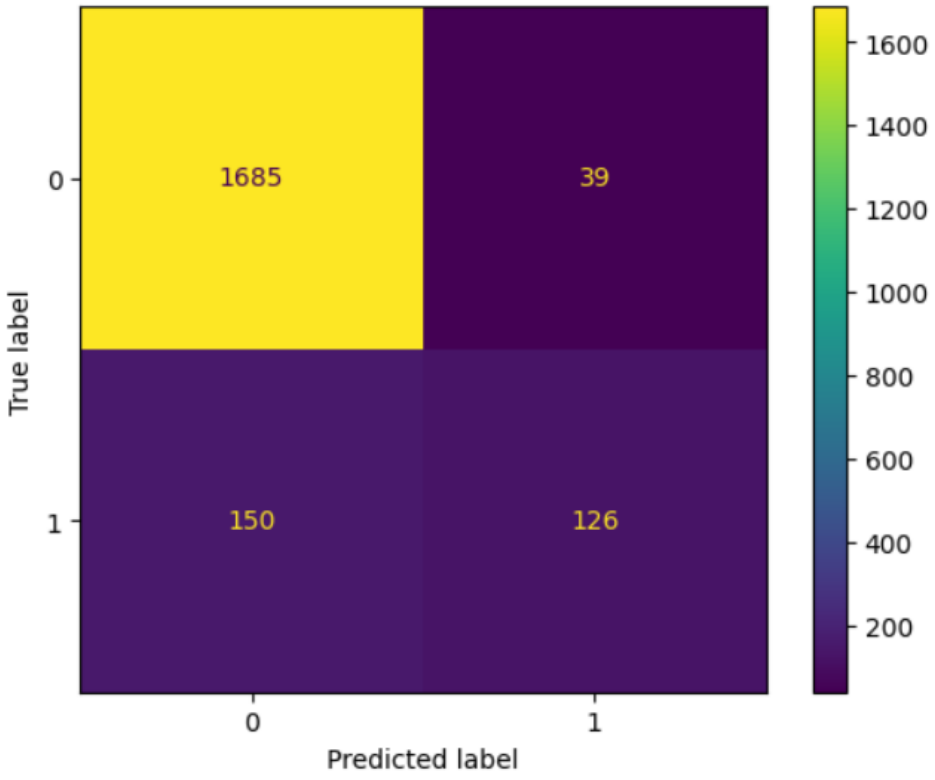
mai mare sau egală cu 3), dar deoarece modelele erau specializate mai mult pe recunoașterea clasei 0, la final nu ajungeam să am mai mult de 500 de 1 în output, ceea ce indica, după părerea mea, că predicțiile nu sunt foarte precise. Așadar, am alternat între condițiile de salvare ale unui model și numărul de 1 din output până ajungeam la un număr satisfăcător. Submisia cea mai bună de la final a fost generată de un model care salva instanțe care clasificau corect cel puțin 100 de anomalii și care aveau o acuratețe de peste 0.9. La final, pentru combinarea predicțiilor celor 5 modele, daca o imagine a fost votată cel puțin o data ca aparținând clasei 1, atunci acolo era încadrată.

Scorurile celor 5 modele salvate au fost după cum urmează:

[0.902999997138977, 0.9020000100135803, 0.906000018119812, 0.9004999995231628, 0.9024999737739563]

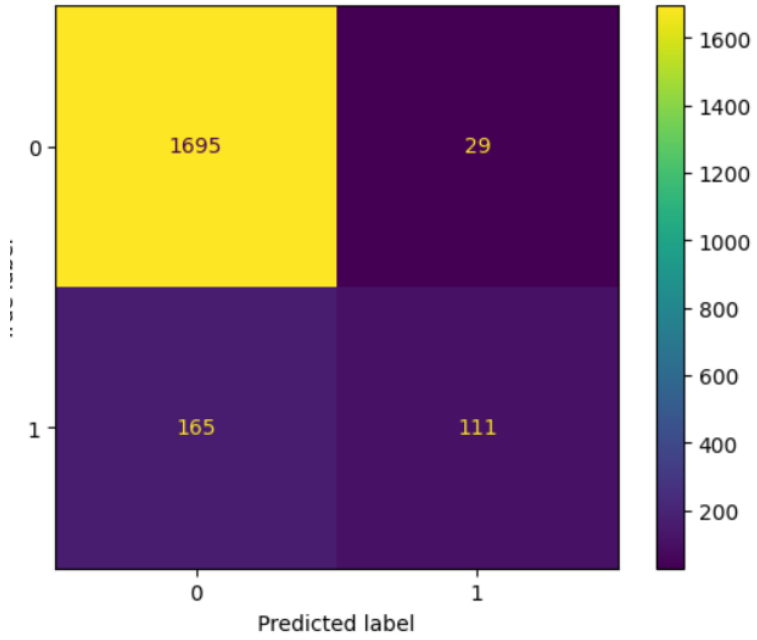
Matricile de confuzie pentru cele 5 modele, scorurile de recall și precision:

1.



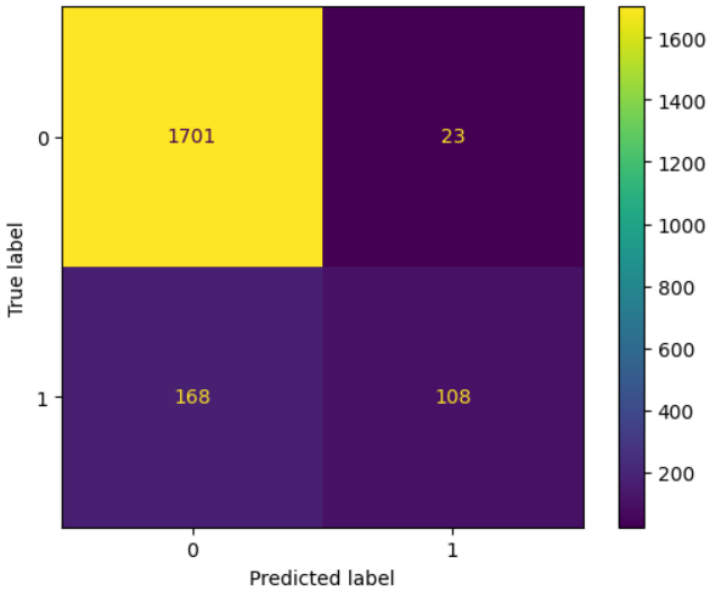

| Class | Precision | Recall |
|---|---|---|
| 0 | 0.9182 | 0.9773 |
| 1 | 0.7636 | 0.4565 |

2.



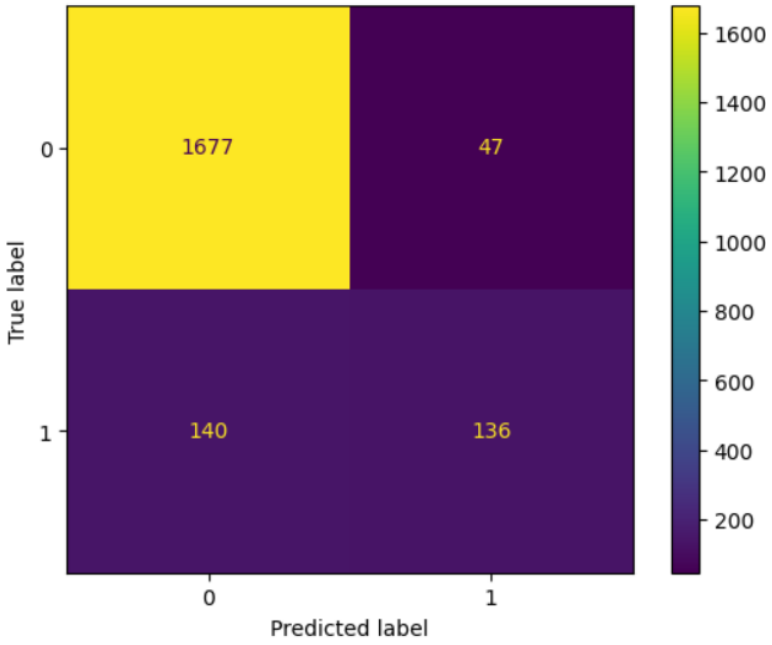

| Class | Precision | Recall |
|---|---|---|
| 0 | 0.9112 | 0.9831 |
| 1 | 0.7928 | 0.4021 |

3.



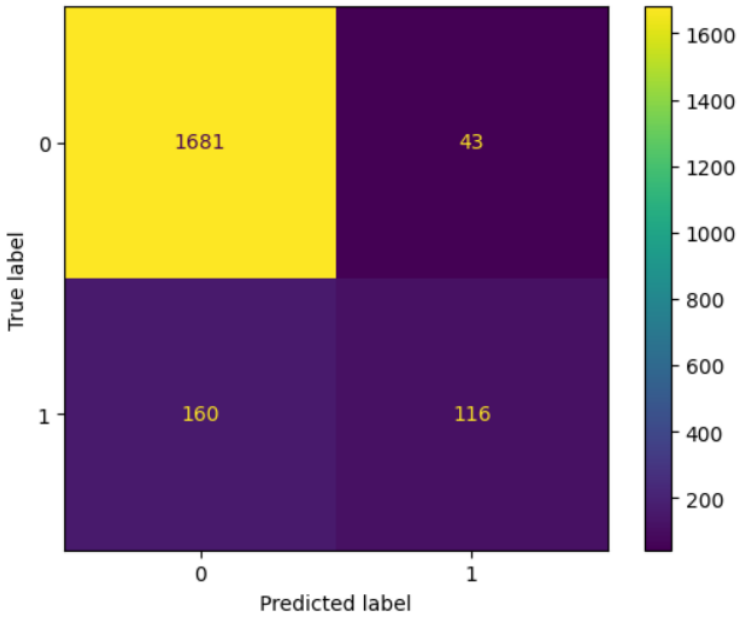

| Class | Precision | Recall |
|---|---|---|
| 0 | 0.9101 | 0.9866 |
| 1 | 0.8244 | 0.3913 |

4.




| Class | Precision | Recall |
|---|---|---|
| 0 | 0.9229 | 0.9723 |
| 1 | 0.7431 | 0.4927 |

5.



| Class | Precision | Recall |
|---|---|---|
| 0 | 0.913 | 0.9750 |
| 1 | 0.7295 | 0.4202 |

## Referințe


- https://towardsdatascience.com/a-simple-cnn-multi-image-classifier-31c463324fa
- https://towardsdatascience.com/increase-the-accuracy-of-your-cnn-by-following-these-5-tips-i-learned-from-the-kaggle-community-27227ad39554
- https://www.tensorflow.org/