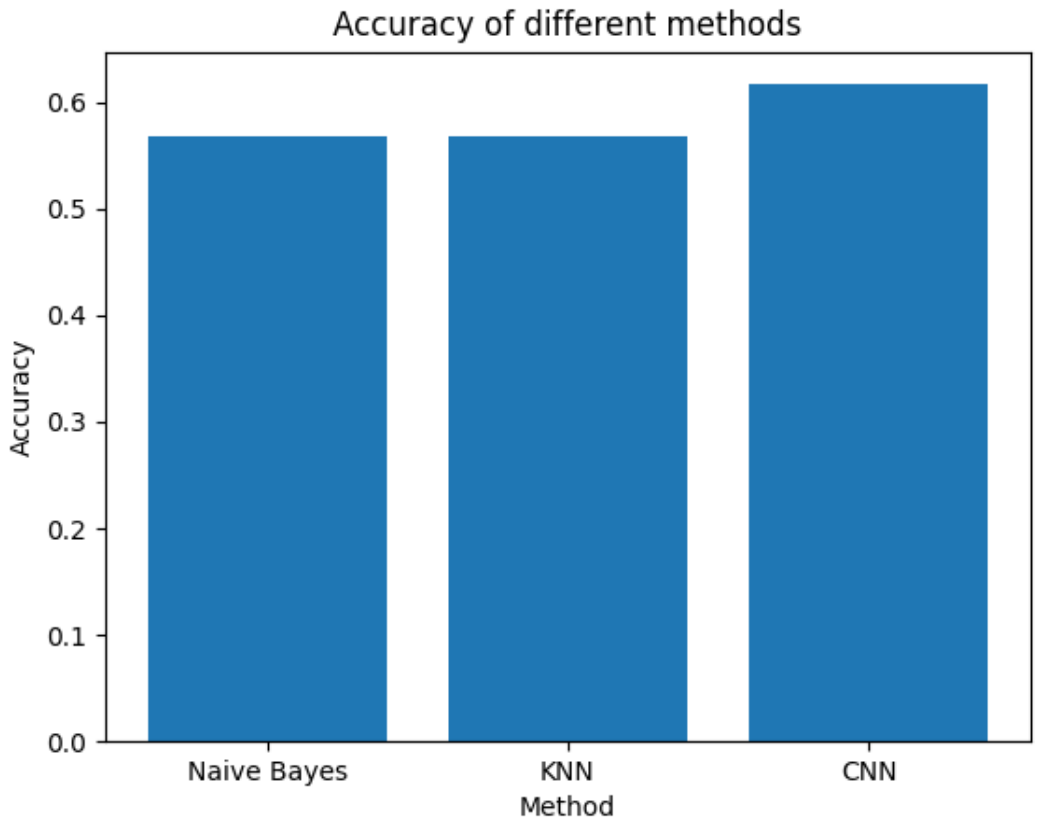
Militaru Mihai-Alexandru

Grupa 243

Proiect Inteligenta Artifciala – Brain Anomaly Detection

Pentru problema de clasificare data, am utilizat trei modele diferite de clasificare, si anume Naïve-Bayes, KNN si CNN. Astfel, am obtinut rezultate diferite pe datele de validare si am incercat sa le imbunatatesc. Modelele de antrenare si procesare a datelor sunt descrise mai jos. Urmatorul grafic ilustreaza acuratetea obtinuta pe datele de validare pe masura ce am folosit cele trei modele.



Procesarea seturilor de date

Pentru a citi seturile de date am folosit atat functia Image.open, din libraria PIL pentru modelele de KNN si Naïve-Bayes, cat si functia cv2.imread din libraria cv2. In functie de modelul ales (detaliate mai jos) am salvat fiecare poza sub forma de vector nparray. Pentru KNN si Naïve-Bayes fiecare poza are dimensiunea de 50176, iar in cadrul modelului CNN vectorii au dimensiunea de 128 * 128 * 1.

In plus, pozele au fost transformate in Greyscale pentru a reduce una din dimensiunile lor, folosind functia convert(L) pentru citirea cu Image.open si cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) pentru citirea cu cv2.imread.

Astfel, obtinem un vector cu valorile pixelilor intre 0 si 255. In continuare, am folosit nparray-uri pentru imaginile de antrenare (train_images_normalized), etichetele de antrenare (train_labels), imaginile de validare (validation-images_normalized), etichetele de validare (validation_labels) si imaginile de testare (test_images_normalized). In cadrul modelelor KNN si CNN vectorii numpy de poze au fost convertiti la tipul float64.

Normalizarea si augumentarea datelor

In functie de modelul ales, normalizarea si augumentarea au fost facute astfel:

- Pentru clasificatorul Naïve-Bayes nu am folosit normalizare sau augumentare.
- Pentru clasificatorul KNN, normalizarea a fost una standard. Am calculat media pozelor de antrenare si deviatia standard a acestora. Standardizarea este calculata prin scaderea deviatiei si impartirea la medie pentru fiecare array de poze.
- Pentru clasificatorul CNN, normalizarea a fost realizata impartind fiecare valoare a pixelilor la 255 folosind functia cv2.normalize(image, None, 0, 1.0, cv2.NORM_MINMAX). Astfel, pixelii vor avea valori in intervalul (0,1) ceea ce conduce la o mai buna acuratete. Pentru acest clasificator am augumentat datele de antrenare prin rotirea imaginilor pe verticala si pe orizontala.

Importanta normalizarii pentru clasificatorul KNN reiese din tabelul urmator unde se observa diferenta intre datele de validare atunci cand se foloseste

normalizare si cand datele sunt pastrate in forma initiala. In tabel am folosit valorile date de F1_Score, deoarece acestea au fost apropiate de acuratetea obtinuta pe Kaggle.

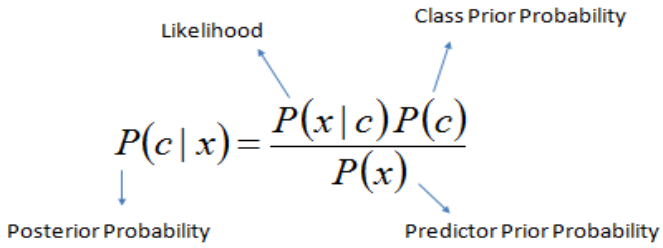
Clasificator	Acuratetea folosind normalizare	Acuratetea fara a folosi normalizare
KNN(3 neighbours)	0.5610262615275167	0.5479310506786906
KNN(5 neighbours)	0.5594176123218937	0.5372348878782713
KNN(7 neighbours)	0.5292678568974285	0.5216051554079723

Clasificatori

Pentru aceasta problema de clasificare am folosit 3 modele cu rezultate diferite: Naïve-Bayes, KNN si CNN.

1. Naïve-Bayes

Clasificatorul Naïve-Bayes se bazeaza pe teorema lui Bayes si calculeaza pentru fiecare imagine daca apartine clasei respective sau nu, fiind o problema de clasificare binara. Astfel, probabilitatea cea mai mare a unei imagini determina daca aceasta apartine clasei respective. Am importat acest model din libraria sklearn.naive_bayes si am folosit modelul MultinomialNB cu parametrii by-default.

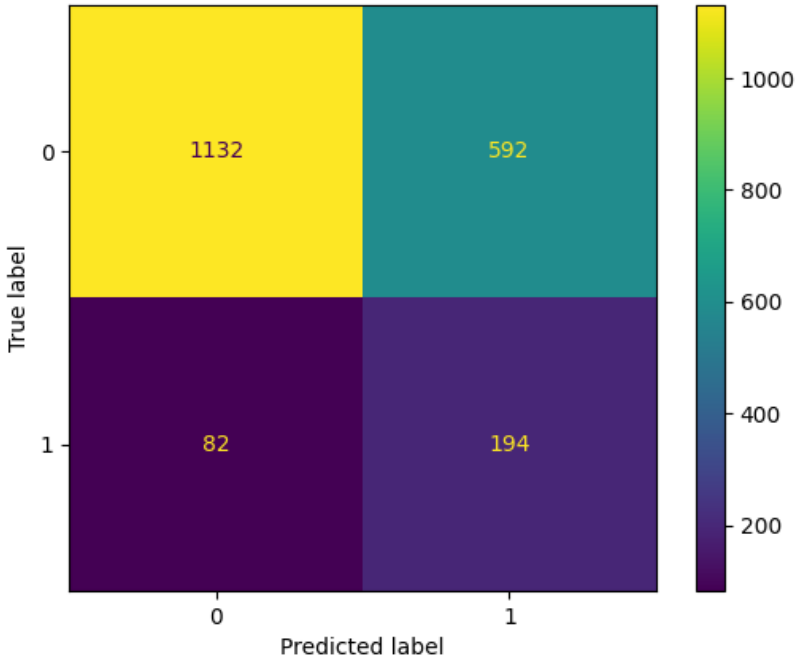


$$P(c \mid \mathrm{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Pentru acest model am important din libraria sklearn f1_score, accuracy_score si recall_score. Valorile obtinute pentru acest model se observa in urmatorul tabel:

	F1 Score	Accuracy Score	Recall Score
Naïve-Bayes	0.5679703194327463	0.663	0.6797555398634789

Mai jos se poate observa matricea de confuzie a acestui model:



2. K Nearest Neighbors (KNN)

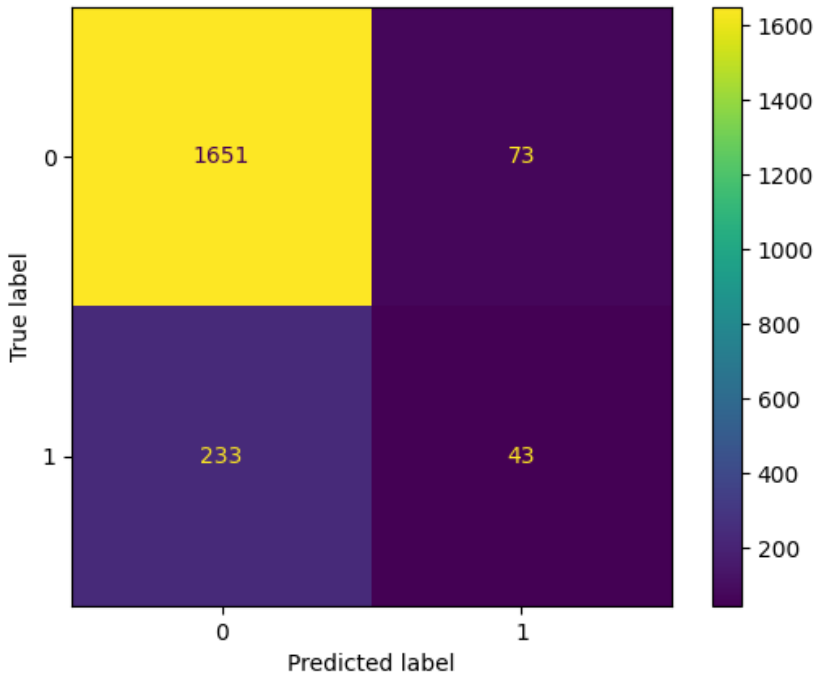
Clasificatorul KNN functioneaza astfel: avand o valoare noua se vor lua in considerare vecinii cu o valoare cat mai apropiata de aceasta, in functie de distanta definita. Normalizarea este importanta pentru acest clasificator, deoarece aceasta este bazata pe distante.

Am importat modelul din libraria sklearn.neighbors, respectiv modelul KNeighborsClassifier cu toti parametrii by-default in afara de n_neighbors. Am testat modelul pentru diverse valori ai parametrului n_neighbors atat pe datele normalizate (cele de antrenare si validare), cat si pe cele nemodificate.

In tabelul de mai jos se observa rezultatele obtinute, calculand f1_score, accuracy_score si recall_score.

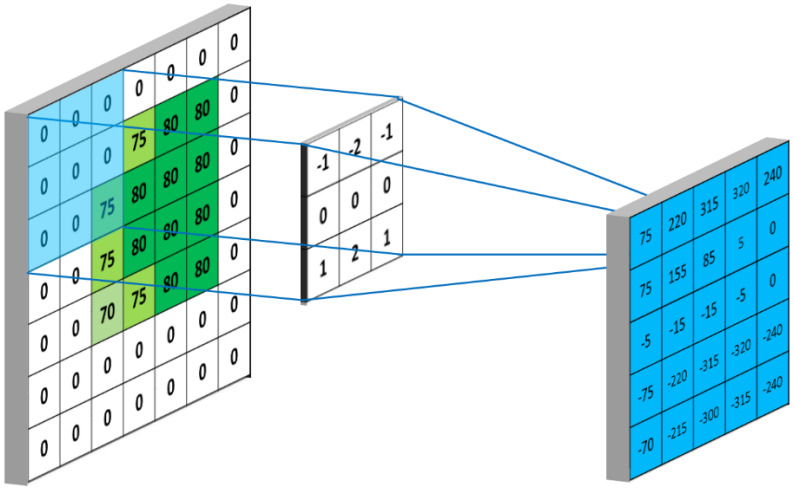
Clasificator	Accuracy Score	F1 Score	Recall Score
KNN (3 neighbors) (fara normalizare)	0.846	0.5479310506786906	0.5424526715760449
KNN (5 neighbors) (fara normalizare)	0.8595	0.5372348878782713	0.5365891590167793
KNN (7 neighbors) (fara normalizare)	0.864	0.5216051554079723	0.5285483708261879
KNN (7 neighbors) (cu normalizare)	0.863	0.5292678568974285	0.5325330374256028
KNN (5 neighbors) (cu normalizare)	0.8655	0.5594176123218937	0.5507204344463499
KNN (3 neighbors) (cu normalizare)	0.847	0.5672881125842798	0.5567268569891388

In final am ales modelul cu datele normalizate ce are ca parametru n_neighbors = 3 avand restul parametrilor by-default, iar matricea de confuzie pentru acest model arata astfel:



3. Convolutional Neural Network (CNN)

Retelele neuronale convolutionale sunt retele neuronale artificiale care folosesc straturi convolutionale ce extrag caracteristici (features) relevante ale imaginilor sau altor tipuri de date cu o structura de tip grila. Aceste caracteristici sunt apoi procesate de straturi de neuroni care au rolul de a clasifica si recunoaste modelele din datele de intrare prin aplicarea unor filtre si harti de atribute.



Convolution Neural Network

Hartile de atribute sunt produse prin aplicarea filtrelor peste imaginea de intrare si sunt trecute printr-o functie de activare. Aceste functii sunt utilizate pentru a introduce non-linearitate in retea, ceea ce permite retelei sa modeleze relatii complexe intre intrarile si iesirile sale. In cadrul retelei mele am folosit functia de activare ReLU (Rectified Linear Unit) si functia Sigmoid.

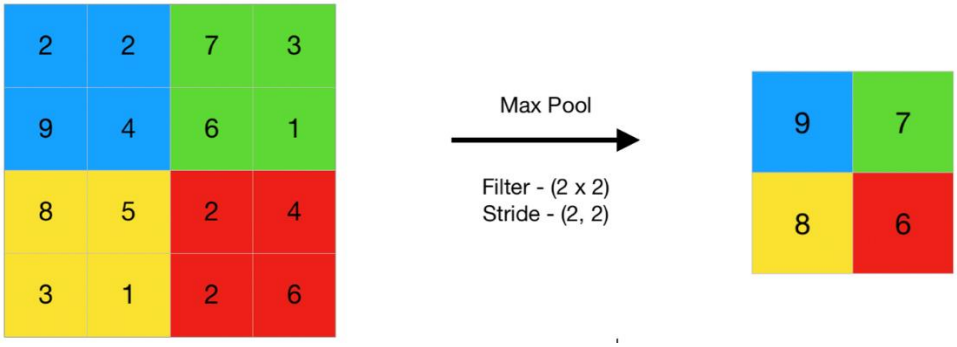
Functia ReLU este definite ca fiind max(0, x) unde x reprezinta iesirea neuronului. Aceasta functie de activare introduce non-linearitatea in retea prin activarea neuronilor cu valori pozitive, iar restul raman inactivi.

Functia Sigmoid este definita in felul urmator:

$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} = 1 - S(-x)$, unde *x* reprezinta iesirea neuronului.

Aceasta functie de activare este utilizata pentru a produce o valoare intre 0 si 1, care poate fi interpretata ca o probabilitate.

Reteaua utilizeaza mai multe tipuri de straturi. Exista un strat de pooling folosit pentru a comprima imaginea si a extrage partile importante din imagine.

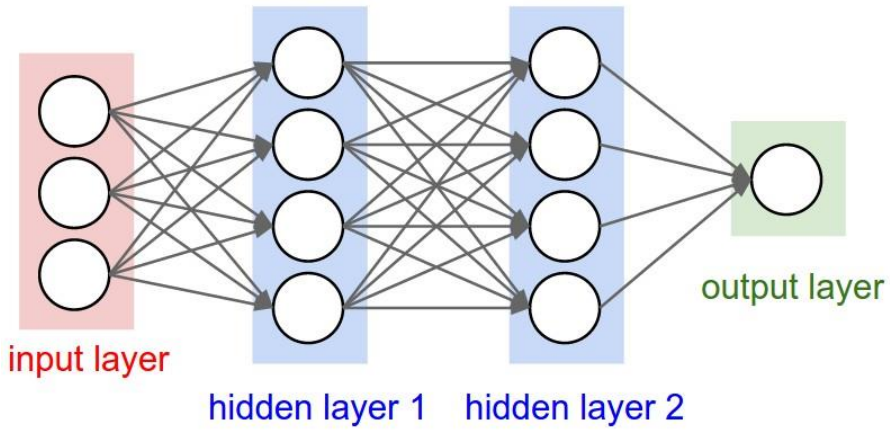


Pooling Layer

Urmatorul tip de strat folosit este cel de dropout pentru a preveni overfitting-ul. Overfitting-ul apare atunci cand modelul se adapteaza prea bine la datele de antrenare si duce la o performanta slaba pe date noi si necunoscute. Functia dropout consta in dezactivarea aleatorie (cu o probabilitate data) a unui procentaj de neuroni din retea in timpul antrenarii.

Stratul de batchnormalization normalizeaza valorile de activare ale fiecarui strat din retea, astfel incat sa aiba medie 0 si deviatia standard 1.

Ultimele straturi primesc o serie de intari si au rolul de a le combina in alte atribute ce vor ajuta la clasificarea datelor.



Convolutional Neural Networks with hidden layers

Ultimul strat contine un neuron, deoarece este o clasificare binara, avand functia de activare sigmoid. Valoarea repsectiva este o probabilitate ce determina

apartenenta la o anumita clasa.

Am utilizat libraria Tensorflow pentru a implementa acest clasificator.

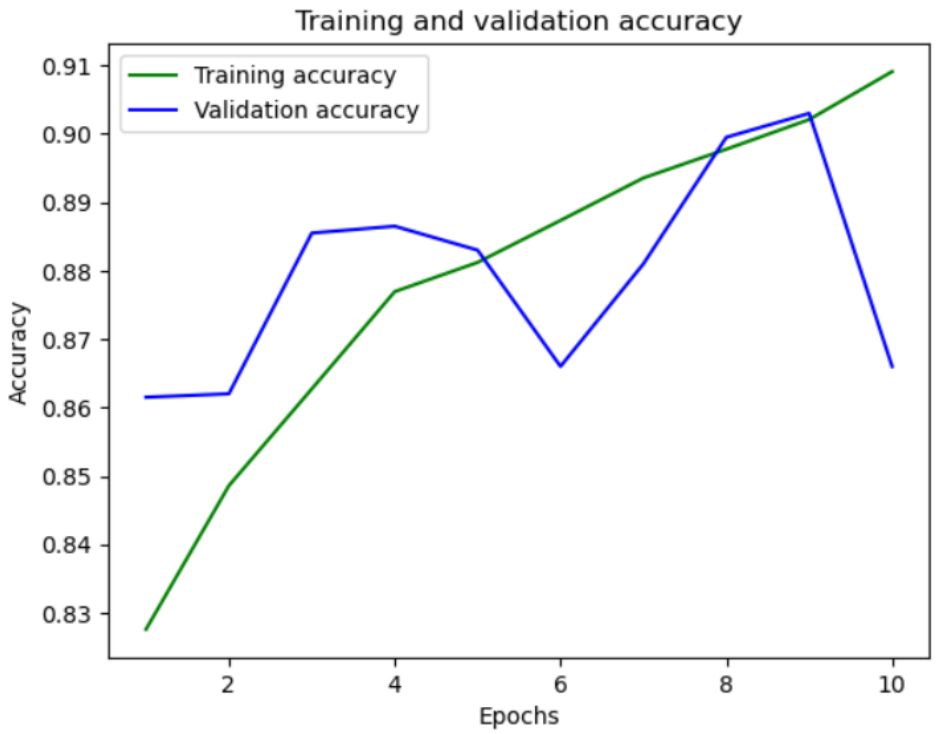
Intr-o prima incercare am utilizat normalizarea pe acest model, impartind valorile pixelilor la 255 si am transformat imaginile in Greyscale. Astfel fiecare imagine este sub forma unui nparray de dimensiunea 224 *224. Valorile pixelilor sunt cuprinse in intervalul (0, 1) pentru o mai buna acuratete.

Aceasta este o prima configurare a modelului CNN:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 224, 224, 32)      320

max_pooling2d (MaxPooling2D  (None, 112, 112, 32)     0
)

conv2d_1 (Conv2D)            (None, 110, 110, 64)      18496

max_pooling2d_1 (MaxPooling  (None, 55, 55, 64)       0
2D)

conv2d_2 (Conv2D)            (None, 53, 53, 128)       73856

max_pooling2d_2 (MaxPooling  (None, 26, 26, 128)      0
2D)

conv2d_3 (Conv2D)            (None, 24, 24, 128)       147584

max_pooling2d_3 (MaxPooling  (None, 12, 12, 128)      0
2D)

flatten (Flatten)            (None, 18432)             0

dense (Dense)                (None, 512)               9437696

batch_normalization (BatchN  (None, 512)              2048
ormalization)

dropout (Dropout)            (None, 512)               0

dense_1 (Dense)              (None, 1)                 513

=================================================================
Total params: 9,680,513
Trainable params: 9,679,489
Non-trainable params: 1,024
_________________________________________________________________
```

Pentru compilarea acestui model am folosit functia de pierdere BinaryCrossenstropy(), deoarece este o clasificare binara, si optimizatorul Adam.

Modelul este antrenat pentru 10 epoci. Parametrul batch_size este setat la valoarea 16, astfel incat sa prelucreze 16 imagini in acelasi timp pe parcursul antrenarii. Am folosit parametrul callbacks pentru a preveni overfitting-ul. Parametrul validation_steps este folosit pentru a calcula numarul de pasi pentru a finaliza o epoca de validare, iar parametrul steps_per_epoch reprezinta numarul de pasi pentru a termina o epoca de antrenare. Evolutia acuratetei pe datele de validare si antrenare se pot vedea in urmatorul grafic:

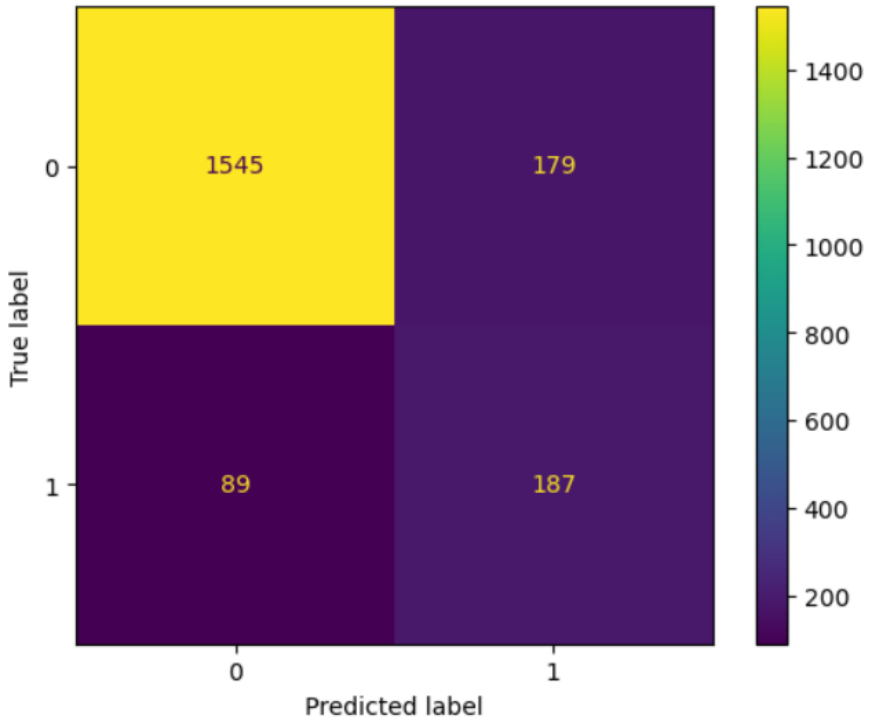


Predictiile sunt sub forma unui vector cu valori cuprinse in intervalul (0,1), cee ace arata probabilitatea fiecarei imagini din setul de testare ca aceasta sa apartine clasei. Drept urmare, valorile sunt rotunjite astfel incat eticheta 0 reprezinta faptul ca imaginea de testare nu apartine clasei, iar eticheta 1 reprezinta apartenenta imaginii la clasa respectiva.

In urmatorul tabel se regasesc F1 Score, Accuracy Score si Recall Score pentru aceasta varianta de CNN:

F1 Score	Accuracy Score	Recall Score
0.5825545171339563	0.866	0.677536231884058

Folosind acesta varianta de clasificator CNN am obtinut urmatoarea matrice de confuzie:



Insa, aceasta varianta a fost imbunatatita atat la nivel de arhitectura a retelei convolutionale, cat si la prelucrarea datelor. Pozele au fost reduse de la dimensiunea de 224*224 la 128*128 folosind functia cv2.resize() din libraria cv2.

Imaginile au fost normalizate la fel ca in varianta precedenta si in plus, au fost augumentate printr-o rotatie pe axa verticala si una pe axa orizontala folosind functia ImageDataGenerator() cu parametrii horizontal_flip si vertical_flip setati la valoarea True. Ulterior am folosit functia flow() pentru a crea un generator de date pentru

antrenament. Generatorul este utilizat pentru a furniza date de antrenament in batch-uri de 32 pentru a incarca 32 de poze simultan. Am folosit functia np.expand_dims din libraria Numpy pentru a extinde imaginea cu o dimensiune pentru a indica faptul ca imaginile sunt in forma Greyscale si au un singur canal.

Pentru aceasta varianta de CNN am marit numarul de straturi, avand urmatoarea configuratie:

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 32)	320
batch_normalization (BatchNormalization)	(None, 128, 128, 32)	128
conv2d_1 (Conv2D)	(None, 126, 126, 32)	9248
batch_normalization_1 (BatchNormalization)	(None, 126, 126, 32)	128
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0
dropout (Dropout)	(None, 63, 63, 32)	0
conv2d_2 (Conv2D)	(None, 61, 61, 64)	18496
batch_normalization_2 (BatchNormalization)	(None, 61, 61, 64)	256
conv2d_3 (Conv2D)	(None, 59, 59, 64)	36928
batch_normalization_3 (BatchNormalization)	(None, 59, 59, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 29, 29, 64)	0
dropout_1 (Dropout)	(None, 29, 29, 64)	0
conv2d_4 (Conv2D)	(None, 27, 27, 128)	73856
batch_normalization_4 (BatchNormalization)	(None, 27, 27, 128)	512
conv2d_5 (Conv2D)	(None, 25, 25, 128)	147584

```
batch_normalization_5 (Batc  (None, 25, 25, 128)      512
hNormalization)

max_pooling2d_2 (MaxPooling  (None, 12, 12, 128)      0
2D)

dropout_2 (Dropout)          (None, 12, 12, 128)      0

conv2d_6 (Conv2D)            (None, 10, 10, 256)      295168

batch_normalization_6 (Batc  (None, 10, 10, 256)      1024
hNormalization)

conv2d_7 (Conv2D)            (None, 8, 8, 256)        590080

batch_normalization_7 (Batc  (None, 8, 8, 256)        1024
hNormalization)

max_pooling2d_3 (MaxPooling  (None, 4, 4, 256)        0
2D)

dropout_3 (Dropout)          (None, 4, 4, 256)        0

flatten (Flatten)            (None, 4096)             0

dense (Dense)                (None, 512)              2097664

dropout_4 (Dropout)          (None, 512)              0

dense_1 (Dense)              (None, 1)                513

=================================================================
Total params: 3,273,697
Trainable params: 3,271,777
Non-trainable params: 1,920
_________________________________________________________________
```

Pentru aceasta varianta de CNN am pastrat aceiasi parametrii la compilarea modelului (optimizagtorul Adam, functia de pierdere BinaryCrossentropy()).

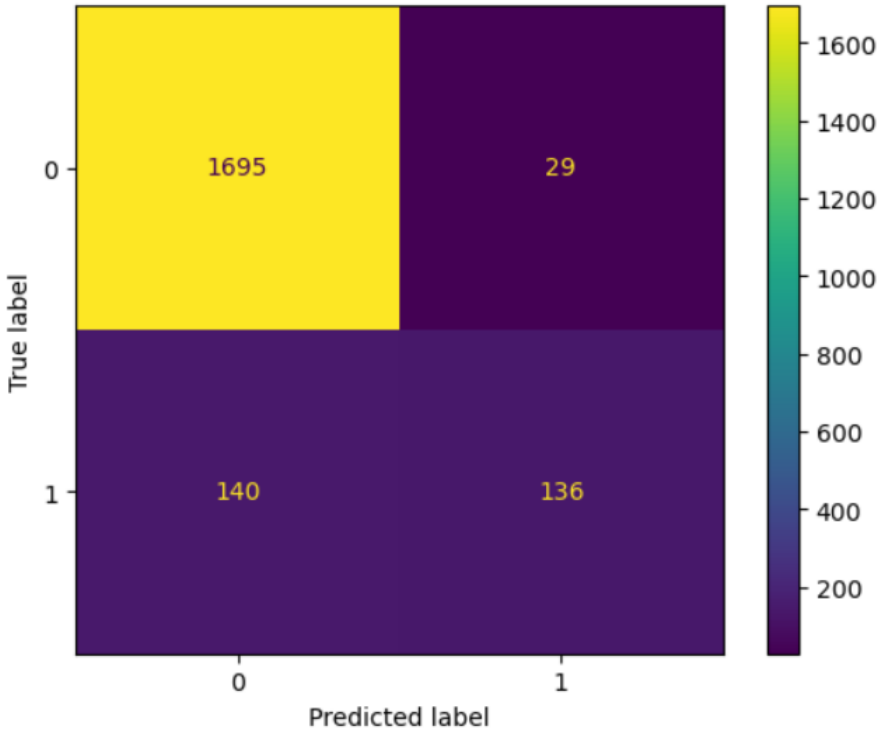
Antrenarea se desfasoara pe parcursul a 75 epoci, insa modelul s-a oprit la epoca 20, datorita functiei Early_Stopping.

Evolutia acuratetei pe datele de validare si antrenare se pot vedea in urmatorul grafic:

In urmatorul tabel se regasesc F1 Score, Accuracy Score si Recall Score pentru aceasta varianta de CNN:

F1 Score	**Accuracy Score**	**Recall Score**
0.6167800453514739	0.9155	0.4927536231884058

Folosind acesta varianta de clasificator CNN am obtinut urmatoarea matrice de confuzie:



Pe aceasta varianta de CNN am obtinut cea mai buna acuratete, atat pe datele de validare, cat si pe cele de 20% din setul de testare de pe Kaggle, acolo obtinand o acuratete de 0.67469.

Ulterior, predictiile au fost rotunjite, deoarece acestea sunt probabilitati si initial au valori in intervalul (0, 1). Rezultatele au fost afisate in fisierul sample_submission.csv. Pentru fiecare imagine este scris numarul ei de ordine si eticheta prezisa.