Crînganu Denis-Florin
Grupa 243

Brain Anomaly Detection

I. Project description

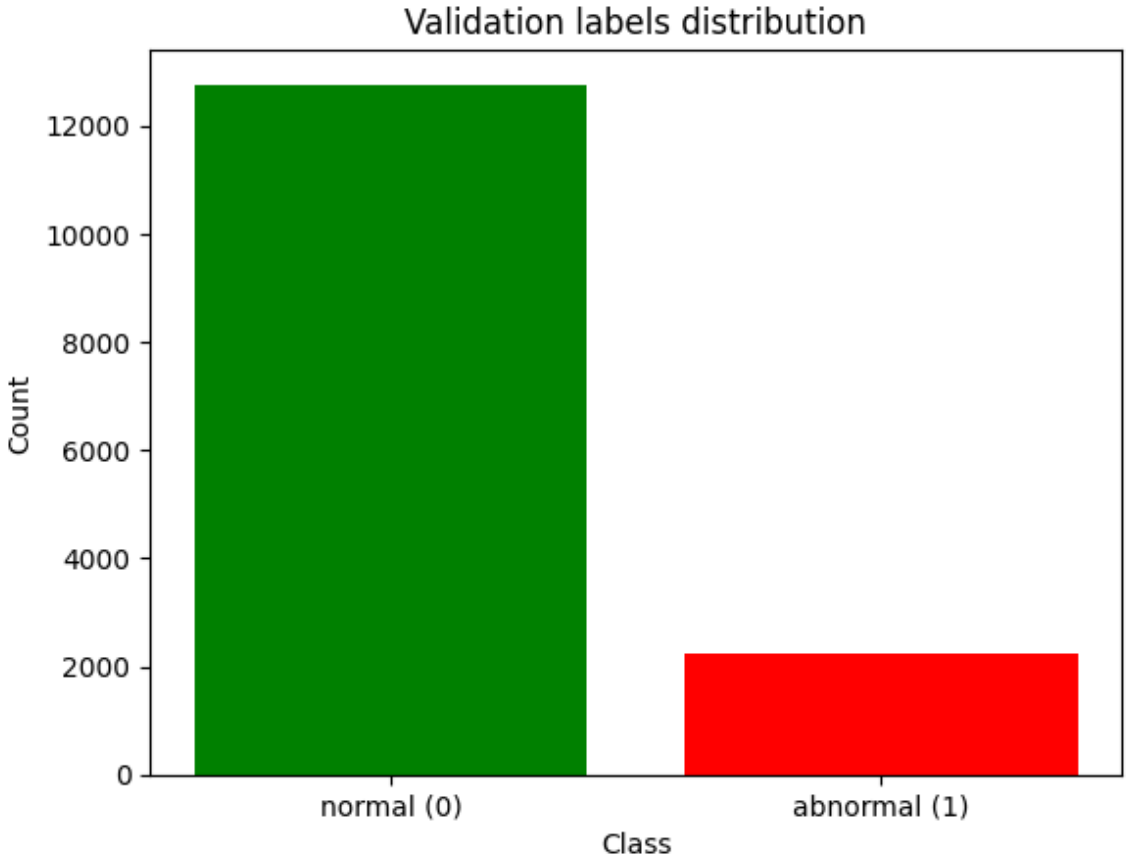
Proiectul presupune clasificarea unor imagini de tip **Brain CT**, având dimensiunea **224x224**, în două clase: **normal (0)** si **abnormal (1)**.

II. Data sets

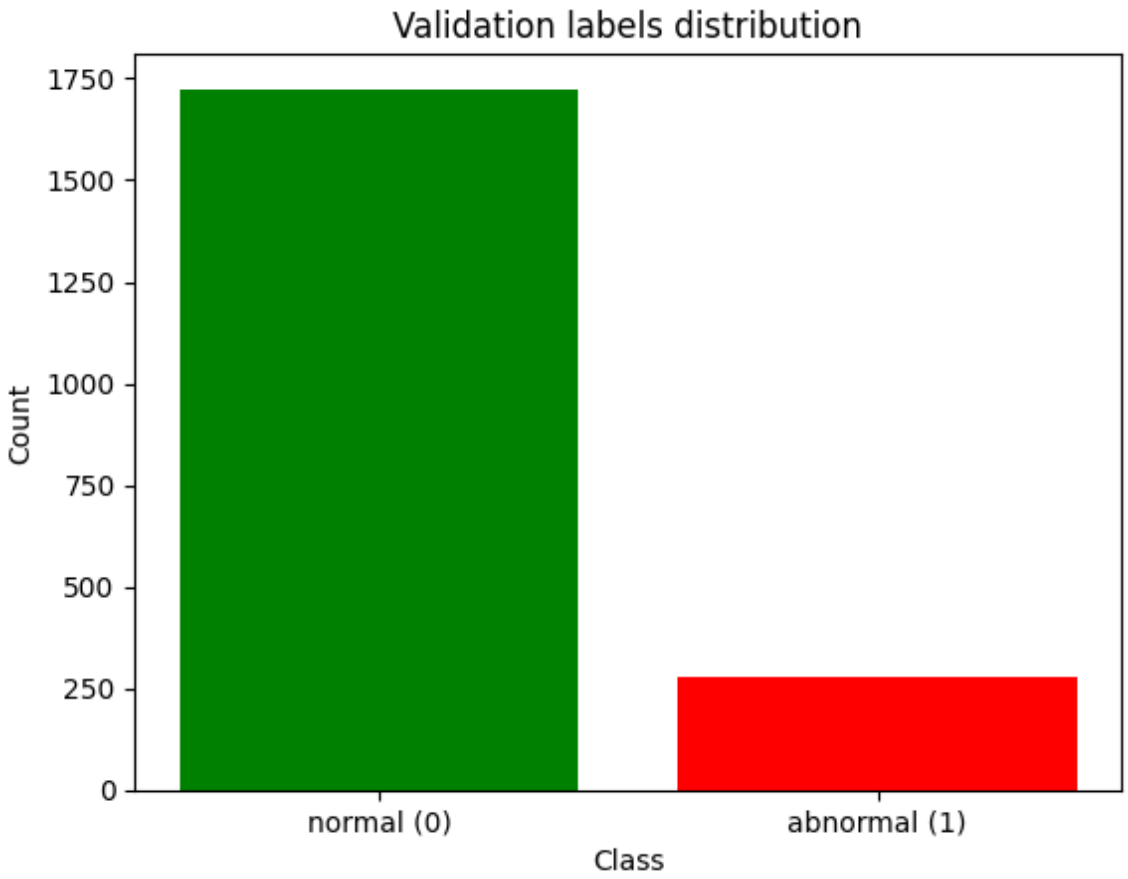
Setul de date primit a fost împărțit în 3 categorii: train, validation și test.

- Setul pentru train este alcătuit in 15000 de imagini însoțite cu labelul corespunzător
- Setul pentru validation este alcătuit din 2000 de imagini însoțite cu labelul corespunzător
- Setul pentru test este alcătuit din 5149 de imagini, pe care modelul trebuia să fac predicțsile finale

Distribuția label-urilor pe setul de train și setul de validation:



Crînganu Denis-Florin
Grupa 243



Din câte se poate observa, distribuția label-urilor este foarte nebalansată, fapt ce ne va crea probleme când vom încerca să clasificăm imaginile folosind un model de ML.

Aceasta problemă poate fi rezolvată prin câteva metode cum ar fi:

- **Undersampling** – ignorăm o parte din imaginile din clasa majoritară astfel încât distribuția claselor să devină echilibrată. Cu toate că această metodă poate merge uneori, este esențial de ținut minte faptul că putem pierde informații foarte utile atunci când ignorăm o parte din imaginile din clasa majoritară;
- **Data augmentation** – aplicăm diferite transformări asupra imaginilor din clasa minoritară, precum: rotation, brightness, shear, elastic, etc, astfel încât numărul imaginilor din clasa minoritară să devină egal cu numărul de imagini din clasa majoritară;
- **Class weights** – metodă prin intermediul căreia îi transmitem modelului faptul că o imagine din clasa X valoarea cât K imagini din clasa Y. Această metodă este folosită pentru modelele de tip CNN, și reprezintă un hiperparametru pentru modul cum se calculează loss function.

Crînganu Denis-Florin
Grupa 243

III. Approaches

Pentru a rezolva această problemă am utilizat 2 algoritmi, cum ar fi:

- **KNN (k-nearest neighbors)**
- **CNN (Convolutional Neural Network)**

IV. KNN

Într-o prima abordare a problemei, am optat pentru folosirea algoritmului KNN (k-neared neighbors). KNN este un algoritm simplu de clasificare ce are rolul de a clasifică puncte reprezentate într-un spațiu, în funcție de primii k cei mai apropiați vecini ai acestuia. Acești k vecini vor participa la un vot care va stabili prin vot majoritar în ce categorie putem încadra elementul pe care vrem să-l clasificăm.

1. Process and data augmentation

Primul pas pentru a face modelul să atingă o performanță cât mai bună a fost să procesez imaginile. Deoarece am observat faptul că imaginile sunt prea mari (original: 224x224), am încercat să micșorez fiecare imagine cu 75%, astfel am adus imaginile la dimensiunea de 56x56 pixels.

Pentru a citi imaginile și a le micșora, m-am folosit de librăria **opencv-python**. Astfel, fiecare imagine era citită folosind **cv2.imread()**, și era transformă într-un **numpy array** cu shape-ul (224, 224), după care micșoram imaginea folosind **cv2.resize()**.

Așa cum am prezentat mai sus, distribuția claselor pe setul de train este foarte nebalansat, fapt ce făcea modelul KNN să clasifice imaginile că normal (0), deoarece această era clasa majoritară. Știind acest lucru, am folosit **undersampling** pentru a echilibra clasele, astfel că selectam 3000 de imagini din clasa normal (0) și 2248 (numărul maxim) de imagini din clasa abnormal (1).

Următorul pas pentru a face modelul să atingă o performanță cât mai bună a fost data augmentation, astfel că pentru fiecare imagine procesată aplicam diferite transformări cum ar fi: **cv2.GaussianBlur()**, ce mă ajută să fac blur pe imagini, respectiv **cv2.add()**, ce mă ajută să fac brightness pe imagini. La final, pentru fiecare imagine îi cream 8 copii, dar cu o transformare diferită.

2. Saving images

Deoarece procesarea și augmentarea imaginilor putea dura până la 5 minute, am decis că să procesez și să augmentez datele, după care să le salvez că **1D numpy arrays** într-un fișier cu extensia **.npy**.

Astfel că imaginile de train după procesare și augmentare le salvam într-un fișier numit "t**rain_data_array_smaller.npy**", imaginile de validation, respectiv test, le dădeam doar resize, fără augmentare, după care le salvăm într-un fișier numit "**validation_data_array_smaller.npy**", respectiv "**test_data_array_smaller.npy**".

Procedând în acest fel, așteptam doar o singură dată că toate imaginile să fie procesate, după care de fiecare dată când doream să testez modelul, încărcam numpy arrays din fișierele salvate. Am decis să salvez imaginile în fișiere cu extensia .npy deoarece scrierea și citirea este foarte rapidă.

Această metodă m-a ajutat atunci când testam hiperparametrii pe modelul KNN, deoarece imaginile rămâneau la fel, doar schimbam hiperparametrii, am reușit să reduc timpul de așteptare de la 5 minute la 15 secunde.

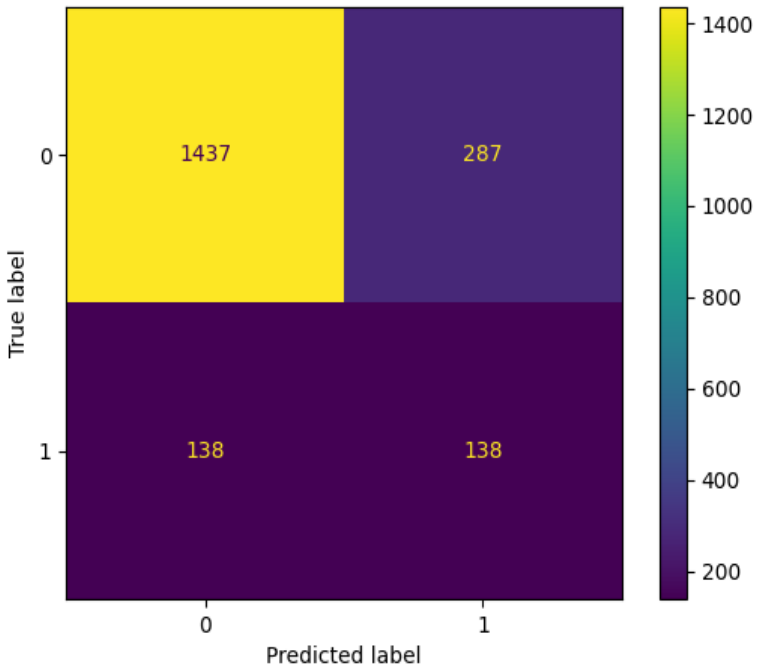
3. Model

Pentru a crea acest model, m-am folosit de librăria **sklearn**, care vine cu modelul deja construit, și fiind nevoie doar să testez hiperparametri și să descopăr optimul pentru rezolvarea acestei probleme.

Metrica pe care am folosit pentru a calcula distanță dintre vecini este **distanța euclidiană**, și am ales ca hiperparametrul **n_neighbors = 21**, în urmă experimentelor efectuate.

4. Result

În urmă executării algoritmului pe setul de validare am obținut **matricea de confuzie**:



Precum si următoarele metrici:

- **Accuracy = 0.7875**
- **Recall = 0.5**
- **Precision = 0.3247**
- **F1_score = 0.39372**

Acest model a reușit să obtină f1_score pe setul de testare egal cu **0.37055 public score,** și **0.41315 private score.**

V. CNN

CNN (Convolutional Neural Network) este o clasă de rețele neuronale artificiale care este utilizată în principal pentru analiza imaginilor și recunoașterea modelelor în datele vizuale. Aceste rețele sunt denumite "**convolutionale**" datorită utilizării operației de convoluție, care este folosită pentru a extrage caracteristici din imaginea de intrare.

Într-un CNN, datele de intrare sunt trecute printr-un set de straturi de convoluție, urmate de straturi de activare și de straturi de reducere a dimensiunii, care reduc dimensiunea datelor înainte de a fi trecute prin alte straturile ulterioare.

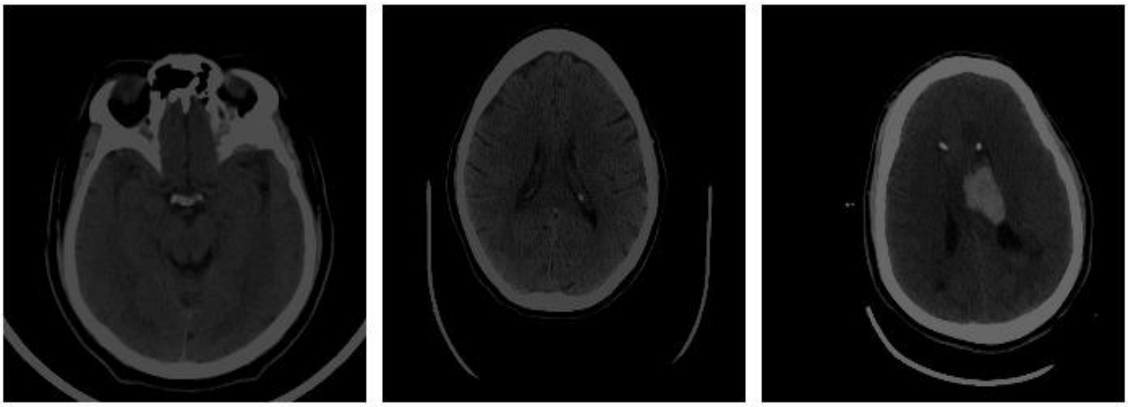
Crînganu Denis-Florin
Grupa 243

1. Image processing

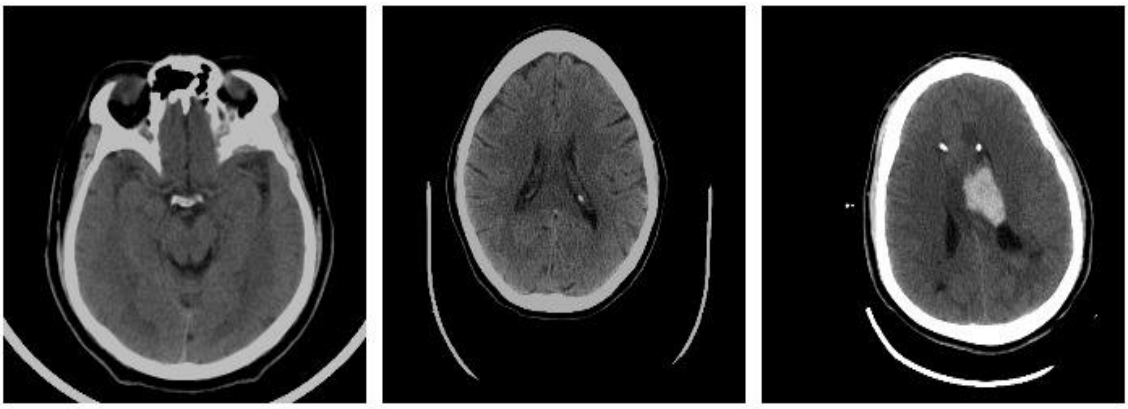
Primul pas pe care l-am făcut pentru a optimiza modelul CNN, a fost procesarea imaginilor și modificarea acestora. La fel ca și pentru KNN, imaginile le-am citit utilizând librăria **opencv-python.**

• Eliminate outliner

Un prim lucru pe care l-am observat la setul de imagini, a fost prezența unui **outliner**, de culoare albă, care se putea găsi în imagini sau nu, sub diferite forme, asa cum se poate vedea în imagine de mai jos:



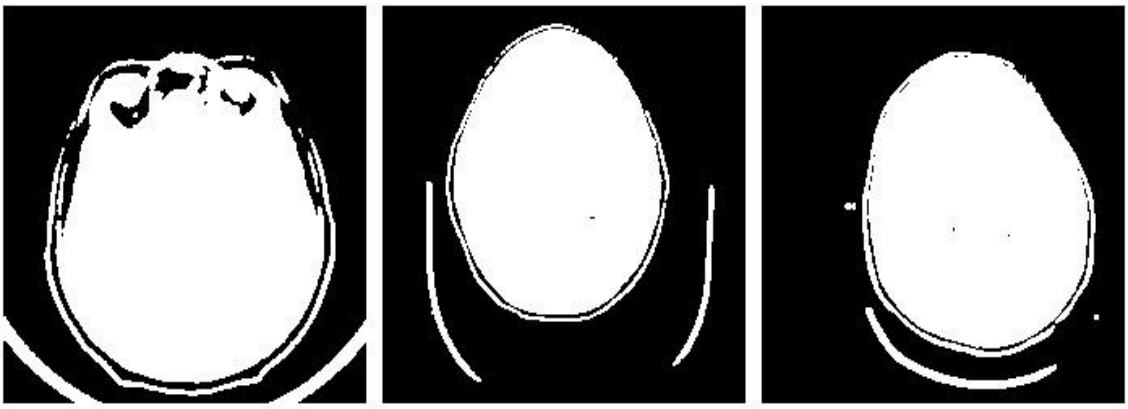
Deoarece am dorit ca CNN-ul să nu fie influențat în alegerea făcută de outliner, am decis să-l elimin. Înainte de a elimina outliner-ul, am crescut contrastul la imagini, astfel că imaginile au devenit:



Deoarece outliner-ul este despărțit de creier, am ajuns la concluzia că pot elimina acest outliner dacă găsesc care este suprafața pe care o acoperă creierul și după care restul pixelilor îi transform în 0.

Ținând cont de acest lucru, următorul pas pe care l-am făcut a fost să transform aceste imagini in **black-white images**, folosind **cv2.threshold()**,

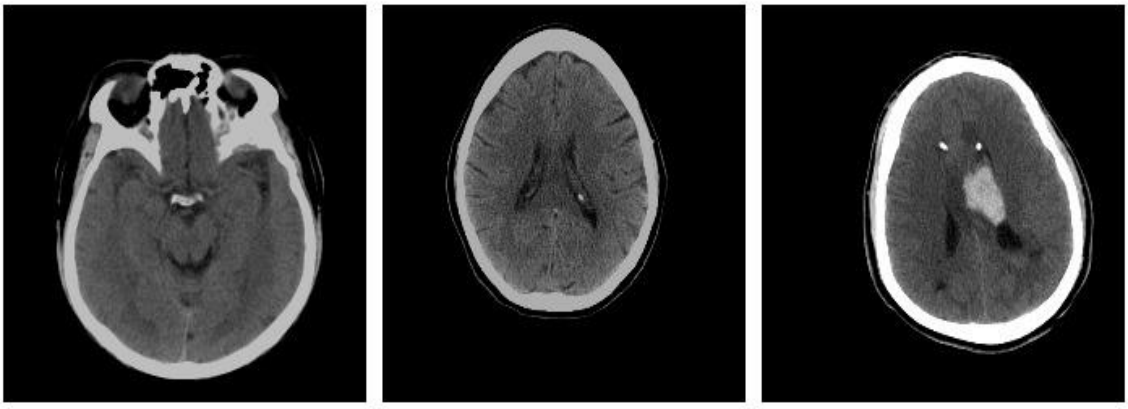
astfel că: dacă valoarea unui pixel depășește o anumită valoare setată de mine, devine alb, altfel devine negru. Aplicând această transformare, imaginile au devenit:



După ce am aplicat această transformare, țineam această formă a imaginilor că o masca pentru imaginea originală, astfel că atunci când eliminim outliner-ul, le voi elimina atât din imagina original cât și din cea sub această formă. Se poate observă că zona creierului o pot afla cu **algoritmul Fill**, astfel:

- Plec de undeva din apropierea mijlocului dintr-un pixel care este alb;
- Merg în fiecare vecin al pixelului curent și-l marchez ca vizitat, vecinul unui pixel se putea afla in N, E, S, V față de poziția pixelului curent;
- Mă opresc când nu mai găsesc niciun astfel de vecin nevizitat.

După ce am marcat fiecare pixel prin care am trecut, voi transforma restul pixelilor pe care nu i-am vizitat în 0, astfel că se vor obține următoarele imagini:

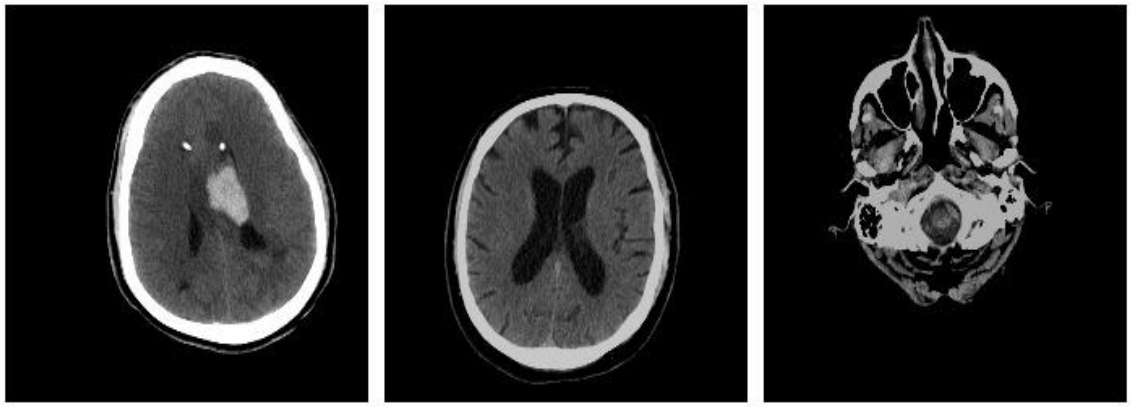


- **Centering images**

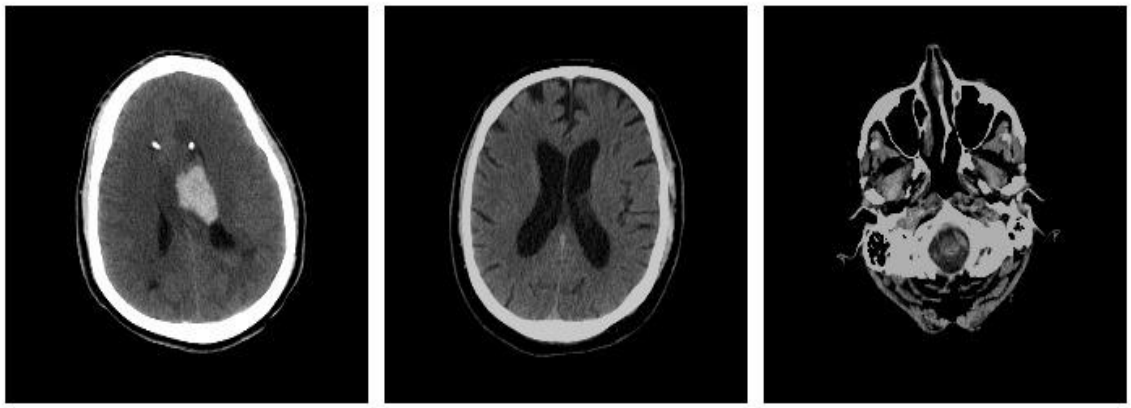
Un al doilea lucru pe care l-am observat la imagini, a fost faptul că aceste nu sunt centrate, lucru ce dacă s-ar întâmpla, ar putea îmbunătăți performanța CNN-ului.

Pentru a putea centra imaginile, m-am folosit de funcția **cv2.moments()** care este utilizată pentru a calcula momentele unei imagini sau ale unei forme dintr-o imagine. Aceste momente reprezintă caracteristici importante ale formei, cum ar fi centrul de masă, suprafața totală sau inerția formei, care pot fi utilizate pentru a efectua operații de analiză și prelucrare a imaginii.

După ce am extras **centrul de masă** al imagini, m-am folosit de funcția **cv2.warpAffine()** pentru a translata imaginea și a o centra, astfel imagini care înainte arătau sub formă:



Au fost transformate în imagini care arătau:



- **Rotating images**

Crînganu Denis-Florin
Grupa 243

Un al treilea lucru pe care l-am observat la imagini, a fost faptul că unele imagini erau puțin rotite, fapt ce putea face ca modelul CNN să nu funcționeze optim.

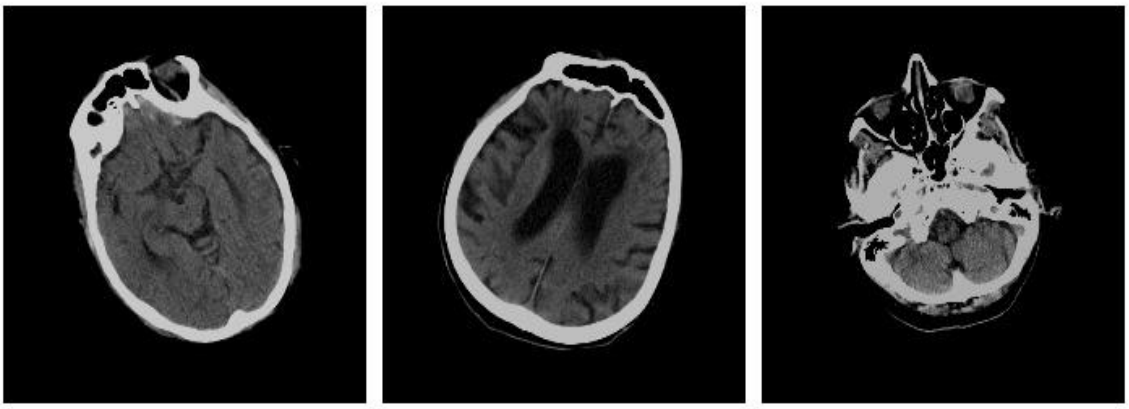
Scopul meu a fost să fac că toate imaginile să fie rotite astfel încât să prezinte o simetrie, ce se află în jumătatea stânga cu ce se află în jumătatea dreapta.

Pentru a face acest lucru să se întâmple, avem nevoie să știm care este unghiul de rotație a creierului din imagine, după care ne putem folosi de funcția **cv2.warpAffine()** pentru a roti imaginea.

Printre încercările mele de a află unghiul de rotație, am observat faptul că aș putea reprezenta zona de interes a imaginii (creierul) sub forma unei elipse, astfel că aș putea desena elipse la diferite unghiuri centrate în mijlocul imaginii, după care să aleg elipsă care are cea mai mare suprafață comună cu zona mea de interes a imaginii.

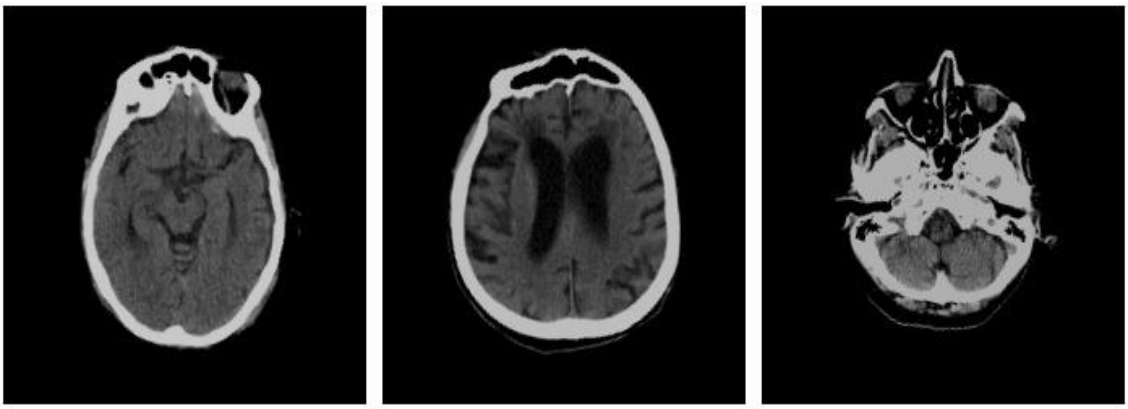
Având acest informații, m-am folosit de funcția **cv2.findContours ()**, care extrage conturul creierului, după care am folosit funcția **cv2.fitEllipse()**, care încearcă să deseneze o elipsă în imagine astfel încât să acopere zona de interes, iar această funcție returnează elipsă creată, dar și unghiul de rotire al acesteia.

Astfel că după ce am aflat unghiul de rotire, imagini care arătau înainte:



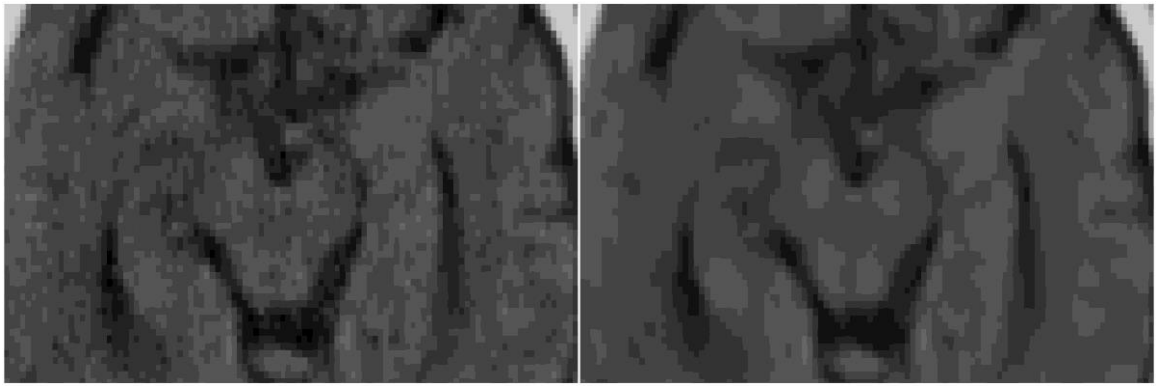
Au fost transformate în imagini care arătau:

Crînganu Denis-Florin
Grupa 243



- **Eliminate noise**

Un al patrulea lucru pe care l-am observat la imagini, a fost prezența unui noise, fapt ce ar fi putut face CNN-ul să ia anumite decizii greșite. Pentru a elimina acest noise m-am folosit de funcția cv2.fastNlMeansDenoising(), astfel imaginile au devenit (stânga original, dreapta fără noise):



- **Edges cases**

În momentul când încercam să aflu unghiul de rotire al imaginii, exista probabilitatea că acesta să fie calculat greșit, astfel că apăreau anumite edge case-uri. Din cauza că numărul acestor edge case-uri era foarte mic, aproximativ 1 la 500 de imagini, iar diferențele erau nu foarte mari de cele originale, am păstrat imaginile cu aceste transformări.

2. Imbalanced data

Cum setul de date de train era foarte nebalansat, atunci modelul CNN tindea să clasifice multe imagini ca False Negative, fapt ce ducea la un f1_score mic. Această problemă poate fi rezolvată prin:

- ## Undersampling

Cum clasa normal (0) era clasa majoritară, cu un număr de 12752 de imagini, în timp ce clasa abnormal (1) avea doar un număr de 2248 de imagini, alegerea numărului de imagini din clasa normal (0) a devenit un hiperparametru pe care am reușit să-l aflu prin mai multe experimente:

- Dacă numărul de imagini din clasa normal (0) este cât mai aproape de numărul de imagini din clasa abnormal (1), atunci modelul tindea să clasifice multe imagini ca **False Positive,** fapt ce ducea la un f1_score mic;
- Dacă numărul de imagini din clasa normal (0) este cât mai aproape de numărul inițial, atunci modelul tindea să clasifice multe imagini ca **False Negative**, fapt ce ducea la un f1_score mic.

Optimul pe care am putut să-l găsesc a fost de 6000 de imagini din clasa normal (0) și 2248 de imagini din clasa abnormal (1).

- ## Data augmentation

O a doua metodă pe care am abordat-o pentru a echilibra datele, a fost data augmentation pe clasa minoritară, în cazul nostru pe clasa abnormal (1).

Această metodă s-a dovedit a fi mai ineficientă decât **undersampling** și am renunțat să experimentez mai mult cu această metodă.

- ## Class weights

Class weights s-a dovedit a fi cea mai optimă metodă, deoarece undersampling implică să renunțăm la anumite date care ar putea fi foarte importante, class weights vine cu altă abordare, și anume: păstrează toate clasele nebalansate, dar transmite modelului faptul că o imagine din **clasa X** valoarea cât **K imagini** din **clasa Y**, fapt ce ajută când se calculează **loss function-ul**.

Asemănător cu **undersampling**, când vine vorba de hiperparametrii, am observat următoarele:

- Dacă valoarea de pe clasa abnormal (1) este cât mai aproape de valoarea de pe clasa normal (0), atunci modelul tindea să clasifice multe imagini ca **False Negative**, fapt ce ducea la un f1_score mic;
- Dacă valoarea de pe clasa abnormal (1) este mare comparativ cu valoarea de pe clasa normal (0), atunci modelul tindea să clasifice multe imagini ca **False Positive**, fapt ce ducea la un f1_score mic.

Optimul pe care l-am putut găsi, a fost să-i atribui clasei normal (0) **weight = 1,** în timp ce clasei abnormal (1), i-am atribuit **weight = 1.5**.

3. Data augmentation

După ce balansam setul de date de train prin una din metodele prezentate mai sus, augmentam imaginile pentru a crea copii a imaginilor originale, dar cu câteva transformări cum ar fi: **rotation, horizontal flip, vertical flip, shear, zoom, width shift, height shift**.

Pentru data augmentation m-am folosit de **ImageDataGenerator** din **keras**, fapt ce mi-a permis să am acces la datele augmentate fără a le ține în memoria RAM, astfel că le țineam în batch-uri de dimensiune [16, 32, 64, 80, 90, 96, 104, 128]. De asemenea, în primele versiune de CNN am folosit data augmentation făcut manual, fără ImageDataGenerator.

De-a lungul experimentelor efectuate, am ajuns la concluzia că data augmentation are un rol foarte important în prevenirea overfitting-ului, iar batch-ul optim găsit a fost 90.

4. Normalization

Pentru a normaliza datele, am folosit metode precum:

- Împărțirea fiecărui pixel la 255, astfel încât valorile pentru fiecare imagine să fie cuprinse între [0, 1];
- **Standard Scaler** din librăria **sklearn**, ce calculează **media** și **deviația** pe train, apoi scade din fiecare imagine media și împarte la deviație.

Amândouă dintre aceste metode au dat rezultate bune, dar în final am optat pentru împărțirea fiecărui pixel la 255.

5. Model

Pentru crearea modelul m-am folosit de **Keras** din **Tensorflow**, și anume am creat un model de tip **Sequential**.

Crînganu Denis-Florin
Grupa 243

Layere aplicate:

- **Conv2D** – layer care creează un kernel de convoluție, cu o dimensiune dată, care aplică un anumit număr dat de filtre asupra inputului pentru a genera matrici de features;
- **MaxPool2D** – layer care reduce dimensiunile inputului prin glisare unei ferestre de o dimensiune dată și luând maximul din această;
- **Dropout** – layer ce are rolul de a reduce overfitting-ul prin dezactivarea unui neuron cu o anumită probabilitate dată ca parametru;
- **Flatten** – layer care aplatizează inputul;
- **Dense** – layer care reprezintă un strat dens conectat de neuronal networks.

Hiperparametrii:

- Pentru layerele convoluționale am folosit mai multe filtre din mulțimea {16, 32, 48, 64, 96, 128, 192, 256, 512}, în timp ce dimensiunea pentru kernel pe care am folosit-o de-a lungul proiectului a fost (3, 3), respectiv funcția de activare a fost “relu”. De asemenea, din experimetele efectuate am observat faptul că padding = “same” producea un scor mai mic decât default, adică padding = “valid”;
- Pentru layerele dropout, am încercat mai multe valorile din mulțimea {0.2, 0.25, 0.4, 0.5}, însă optimul la care am ajuns a fost: droput de 0.25 după fiecare layer convolutional, și dropout de 0.5 după fiecare layer dense;
- Pentru layerele dense, am folosit mai multe unități din mulțimea {128, 256, 512, 1024, 2048}, și în același timp mai multe layere de acest timp, iar funcția de activare folosită a fost doar “relu”;
- Ultimul layer dense este alcătuit dintr-o singură unitate și folosește că funcție de activare “sigmoid”.

Compilare:

- Ca funcție de loss, am folosit mai multe funcții precum: **BinaryCrossentropy**, **BinaryFocalCrossentropy**, **CosineSimilarity**, însă cea mai optimă dintre toate dovedindu-se a fi **BinaryCrossentropy**;
- Ca optimizator, am folosind **adam** cu un **learning rate de 0.001**. Pe parcursul proiectului am încercat să folosesc **ReduceLROnPlateau**, dar în final optimul a fost un learning rate constant de 0.001;

- Metricile pe care le-am folosit au fost **acuratețea** și **f1_score**, pe care l-am afișat folosind funcția **F1Score()** din librăria **tensorflow-addons.metrics**.

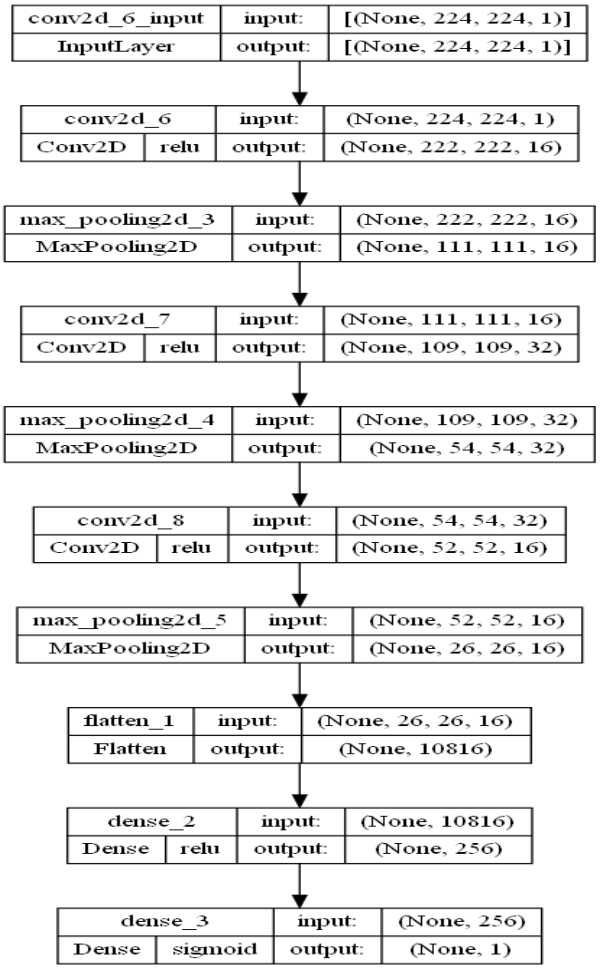
Callbacks:

- Am folosit un callback care să salveze modelul după fiecare epoca (**checkpoints**), indiferent de rezultatele opțiune, astfel încât la final să pot analiza fiecare epoca și să pot alege modelul care consideram că este cel mai bun.

1. First version of CNN

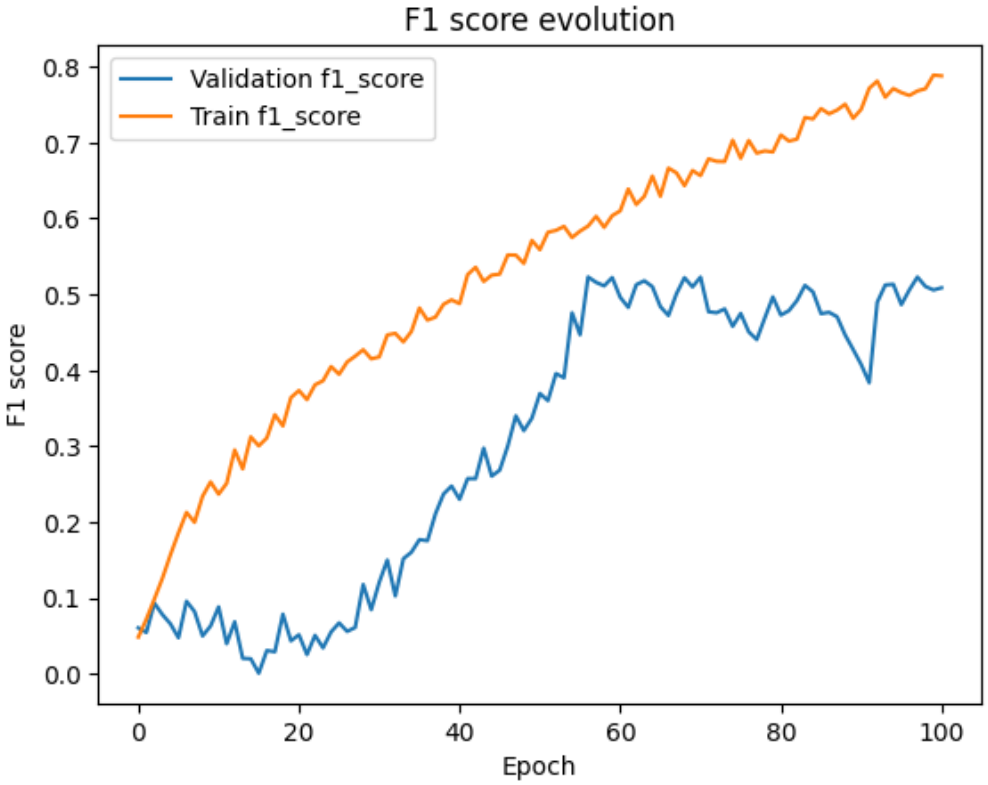
Pentru prima versiune de CNN am folosit **data augmentation** pentru a echilibra clasele, astfel ca pentru imaginile normal (0) aplic doar un singur filtru de brightness, in timp ce pentru imaginile abnormal (1) aplicam mai multe valori de brightness si roteam imaginea la diferite unghiuri. Imaginile erau tinute pe dimensiunea (64, 64) in format **grayscale**.

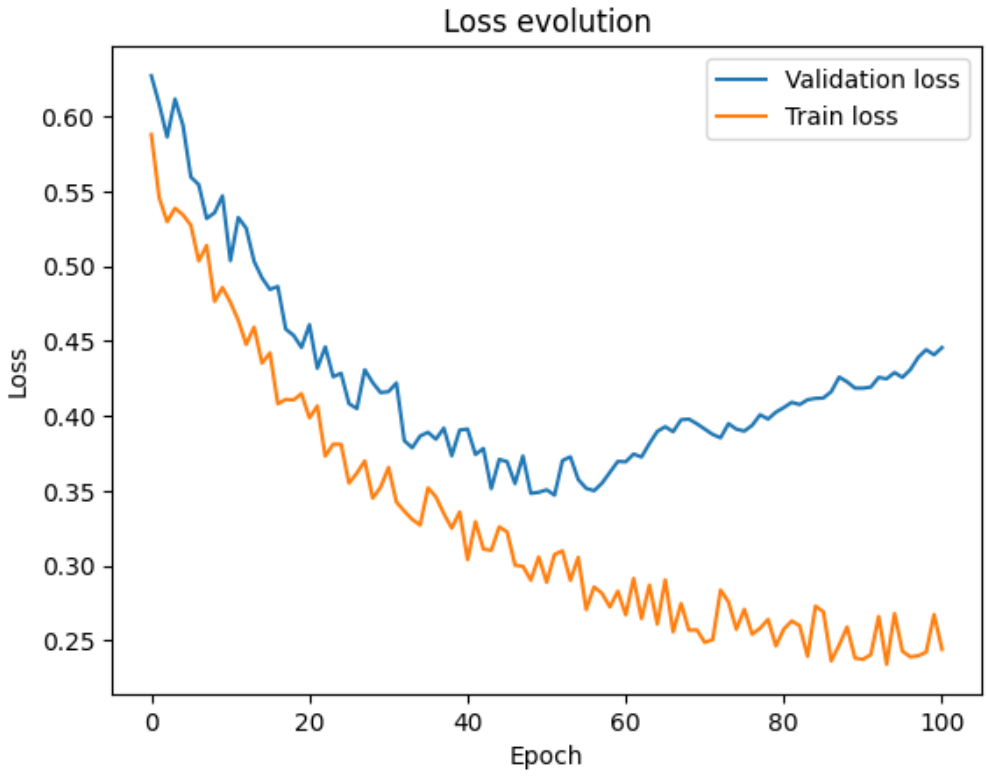
Arhitectura modelului este:



Crînganu Denis-Florin
Grupa 243

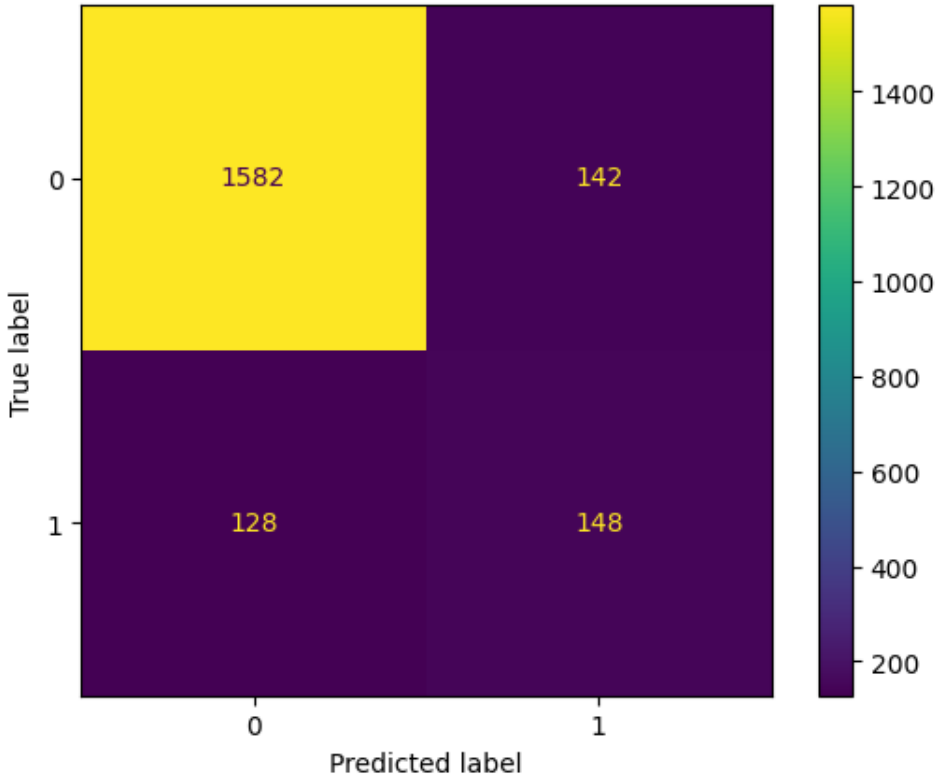
- **F1 score and loss evolution**





- **Result**

Matricea de confuzie pentru cea mai bună epocă al acestui model este:



- **Accuracy = 0.865**
- **Recall = 0.5362**
- **Precision = 0.5103**
- **F1_score = 0.5230**

Acest model a reușit să obțina f1_score pe setul de testare egal cu **0.5302 public score,** si **0.51325 private score.**

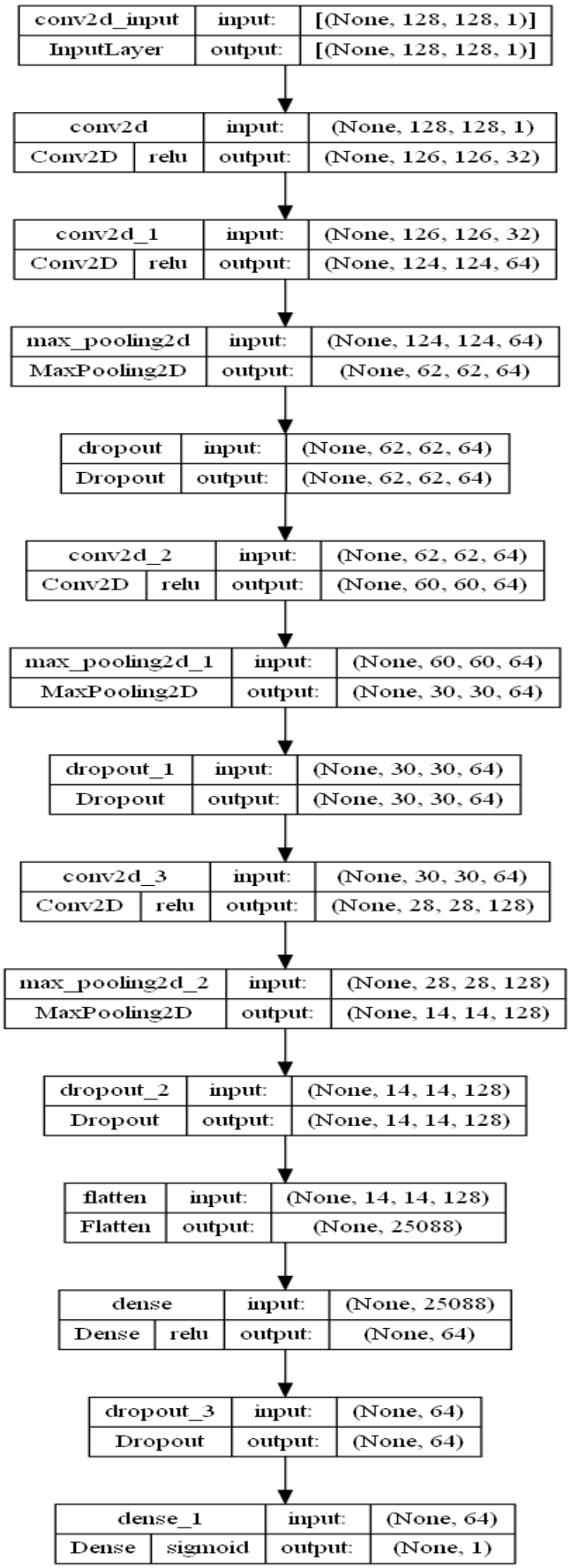
2. Second version of CNN

Pentru cea de-a două versiune de CNN am încercat să creez o rețea mai mare, adăugând mai multe layere convoluționale.

Procesarea datelor am păstrat-o că la prima versiune, am folosit **undersampling** cu o rație de (6500, 2248), iar pentru augmentarea datelor am folosit **brightenss** și **rotation** la diferite unghiuri.

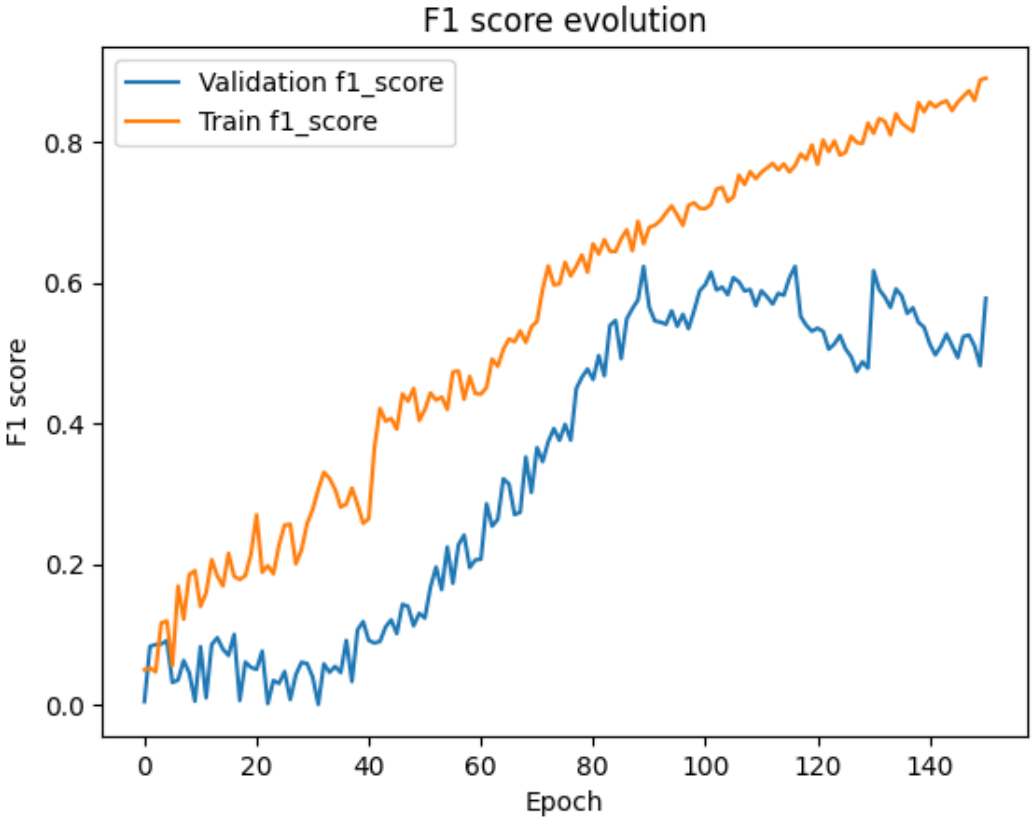
Crînganu Denis-Florin
Grupa 243

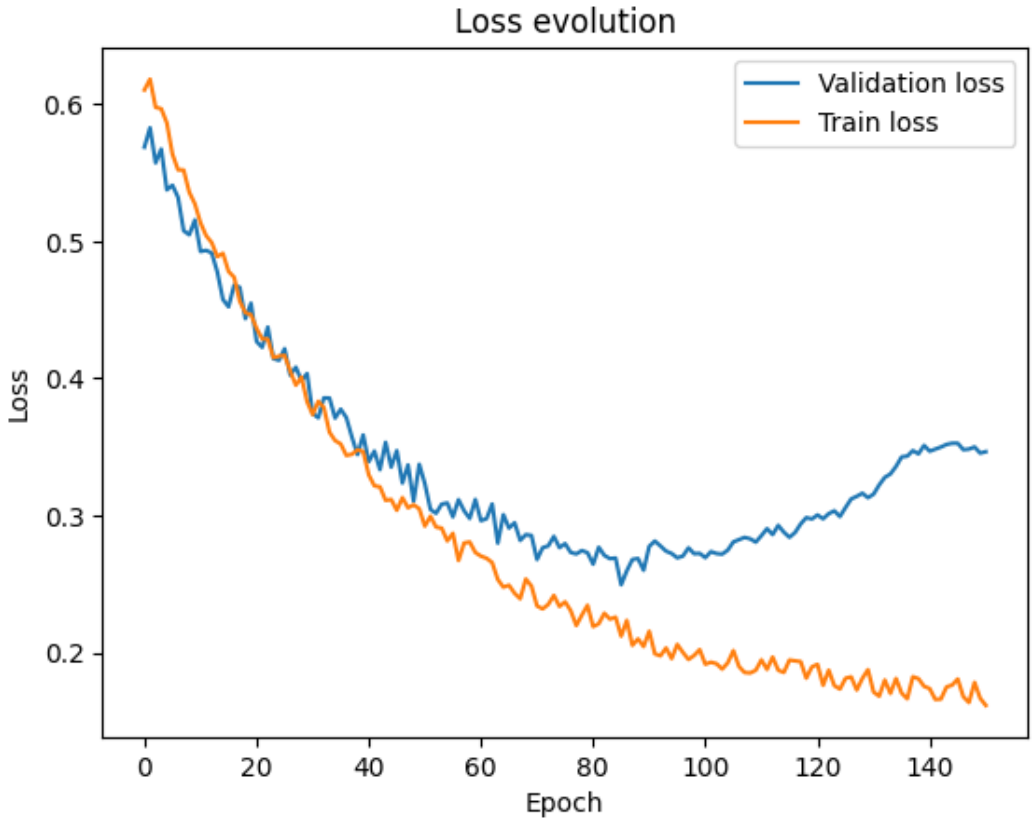
Arhitectura modelului este:



Crînganu Denis-Florin
Grupa 243

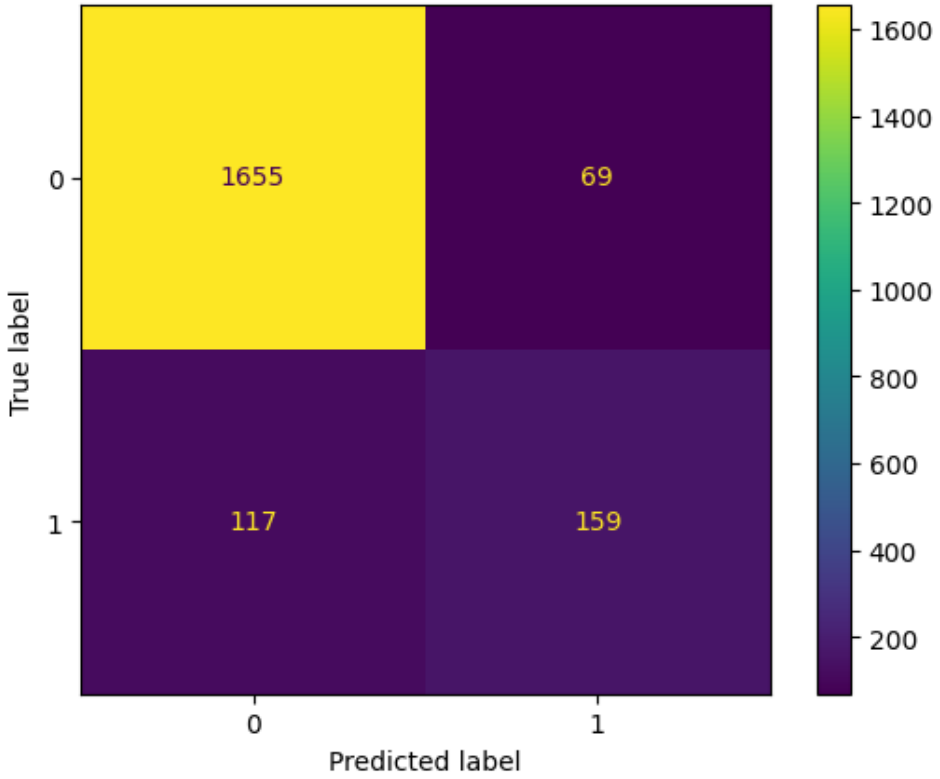
- **F1 score and loss evolution**





- **Result**

Matricea de confuzie pentru cea mai bună epocă al acestui model este:



- **Accuracy = 0.907**
- **Recall = 0.5760**
- **Precision = 0.6973**
- **F1_score = 0.6310**

Acest model a reușit să obțina f1_score pe setul de testare egal cu **0.6400 public score,** si **0.59935 private score.**

3. Final version of CNN

Versiunea finală de CNN față de versiunea anterioară este alcătuită din mai multe layere convoluționale. Mai exact modelul a fost împărțit în blocuri de forma: **Conv2D**, **Conv2D**, **MaxPooling2D**, **Dropout**. Numărul de filtre pentru fiecare Conv2D a fost în ordine **[64, 128, 256]**. După blocul final, urma un layer **Flatten**, un layer **Dense** cu 128 de unități și **output-ul** care este un layer Dense cu o singură unitate.

Fiecare layer convoluțional are un kernel de dimensiune (3, 3), iar pentru MaxPooling2D am folosit dimensiunea (2, 2).

Cu excepția ultimul layer care are ca funcție de activare “**sigmoid**”, restul layerelor au ca functie de activare “**relu**”.

Fiecare MaxPolling2D este urmat de un **Dropout de 0.25**, iar fiecare layer Dense, cu excepția ultimul layer, este urmat de un **Dropout de 0.5**.

Din experimentele efectuate, prin încercarea a mai mult combinații de layere, am ajuns la concluzi că două layere convoluționale urmate de un MaxPooling2D ajută mult la peformanta CNN-ului, comparativ cu un singur layer convoluționat urmat de MaxPooling2D.

De asemenea, Dropout-ul a avut un rol foarte important în prevenirea overfitting-ul, din experimentele efectuate în care am încercat să nu adaug Dropout, rezultatele au fost foarte mari pe setul de validation, însă în realitate, acesta se descurcă slab pe setul de testare.

- ## Images

Imaginile le-am păstrat procesate așa cum am prezentat mai sus, doar că de această dată am decis să țin imaginile pe o rezoluție de **128x128 pixels**, lucru ce s-a dovedit a îmbunătăți mult performanță CNN-ului.

- ## Imbalanced data

Pentru a evita problemei claselor care erau foarte nebalansate, am folosit **class_weights**, astfel după mai multe experimente efectuate am reușit să obțin optimul cu valorile **{0: 1., 1: 1.5}**.

- ## Data Augmentation

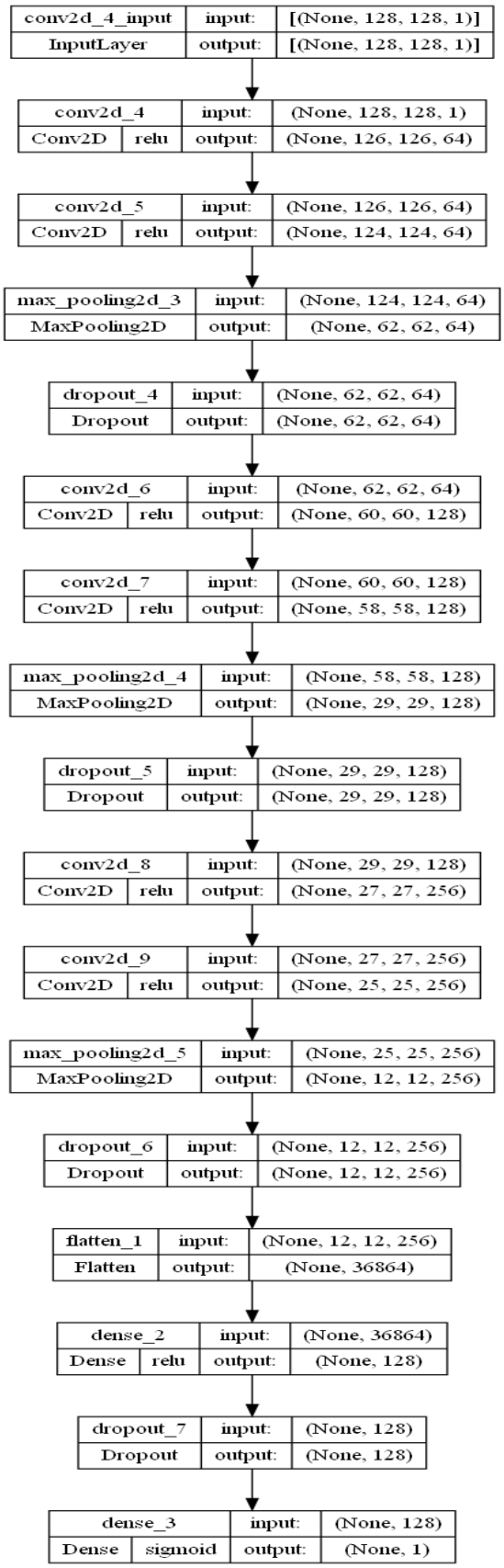
Pentru augmentarea datelor m-am folosit de ImageDataGenerator din keras, astfel că fiecărei imagini îi aplicam: **horizontal flip**, **width shift**, **height shift**, **shear** și **zoom**.

Acest mod de a augmenta datele s-a dovedit a juca un rol foarte important în prevenirea **overfitting-ului**, astfel crescând foarte mult performanța CNN-ului.

Din experimentele efectuate, am observat faptul că **horizontal flip** a fost cea mai bună transformare care se putea aplică, deoarece imaginile prezintă o anume simetrie, CNN-ul ajungea să interprezente o imagine căreia i s-a aplică horizontal flip ca o imagine cu totul nouă.

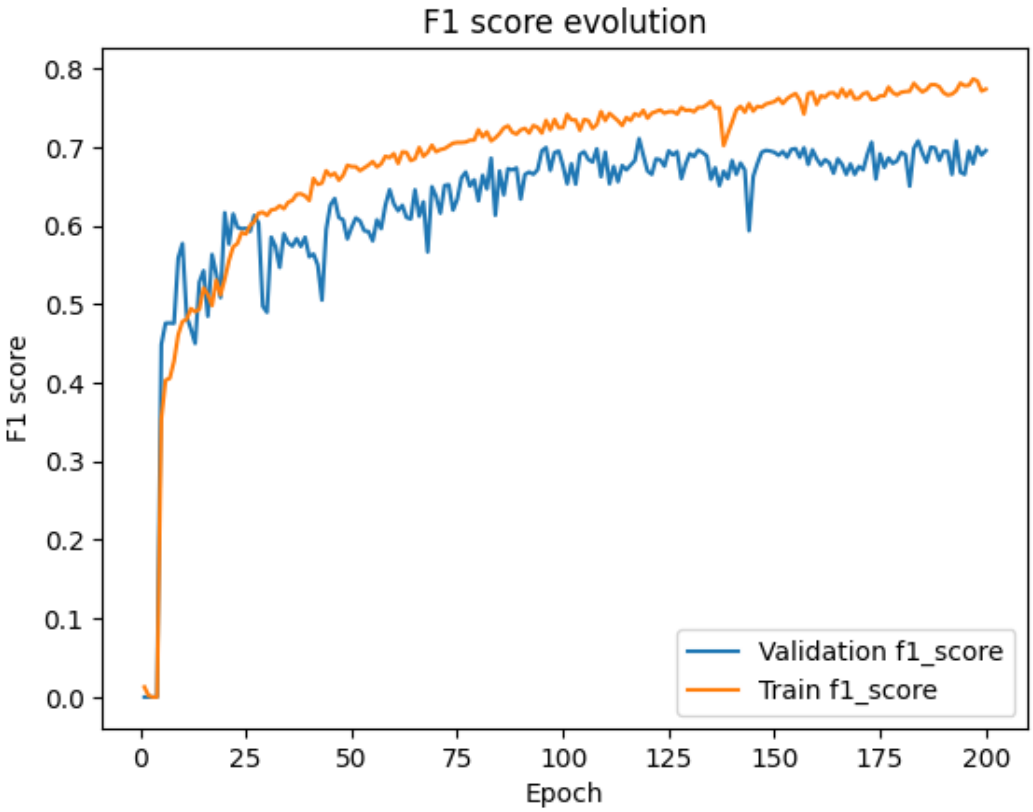
Crînganu Denis-Florin
Grupa 243

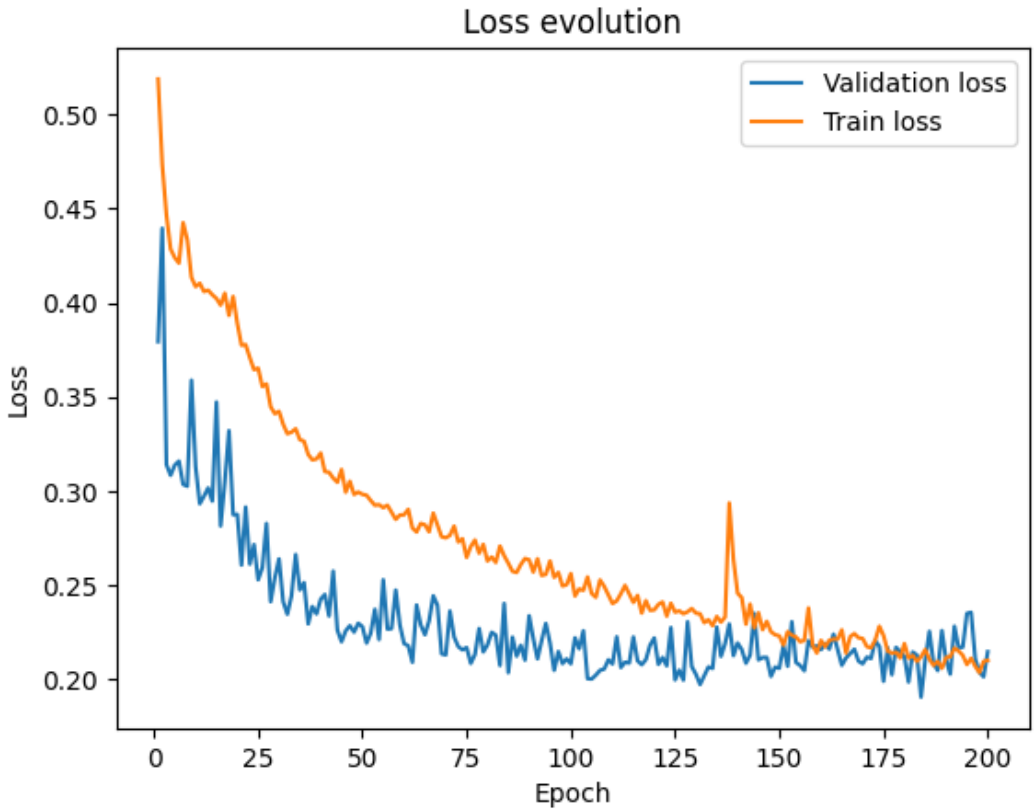
Arhitectura modelului este:



Crînganu Denis-Florin
Grupa 243

- **F1 score and loss evolution**

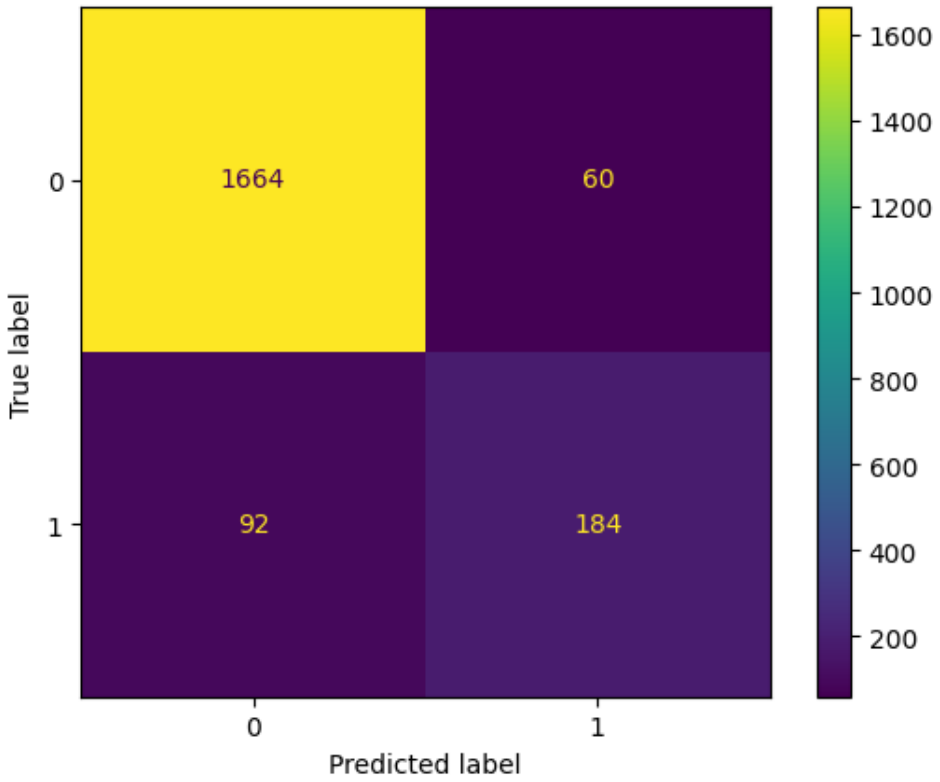




Crînganu Denis-Florin
Grupa 243

- **Result**

  Matricea de confuzie pentru cea mai bună epocă al acestui model este:



- **Accuracy = 0.924**
- **Recall = 0.6666**
- **Precision = 0.7541**
- **F1_score = 0.7076**

Acest model a reușit să obțină f1_score pe setul de testare egal cu **0.74809 public score** și **0.73139 private score**, iar maximul pe private score a fost atins cu altă submisie care a obținut **0.74103 public score** și **0.73694 private score**.