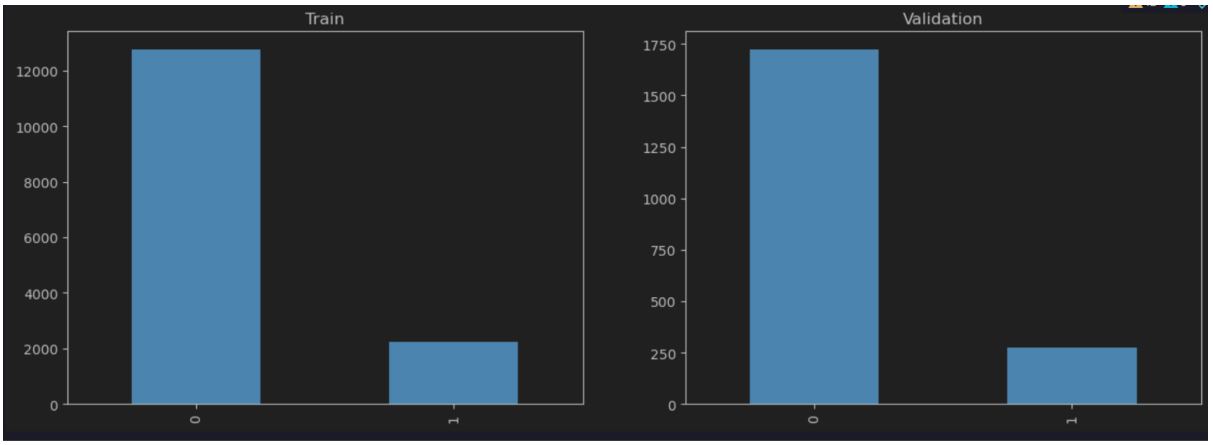
Documentație Proiect ML

Descrierea proiectului

Proiectul presupune clasificarea unor imagini tip computer tomograf în două clase distincte. Clasa 0 pentru imaginile ce nu prezinta anomalii, respectiv clasa 1 pentru cele care prezinta.

Datele de antrenare și validare

Înainte de încercarea oricărui model am analizat datele de antrenare, cât și de validare. Am observat ca datele sunt foarte unbalanced, astfel pentru modele trebuia sa abordez una dintre variantele cunoscute pentru acest tip de date, undersampling, oversampling sau antrenare cu weights precalculate.



Pentru toate modelele am normalizat pixelii între valorile [0,1] împărțind fiecare pixel la 255.

Primul algoritm testat - Random Forest

Random forest este un algoritm de învățare supervizată pentru probleme de clasificare și regresie. Algoritmul se bazează pe o mulțime de arbori de decizie, unde fiecare arbore este antrenat pe o submulțime aleatoare a datelor.

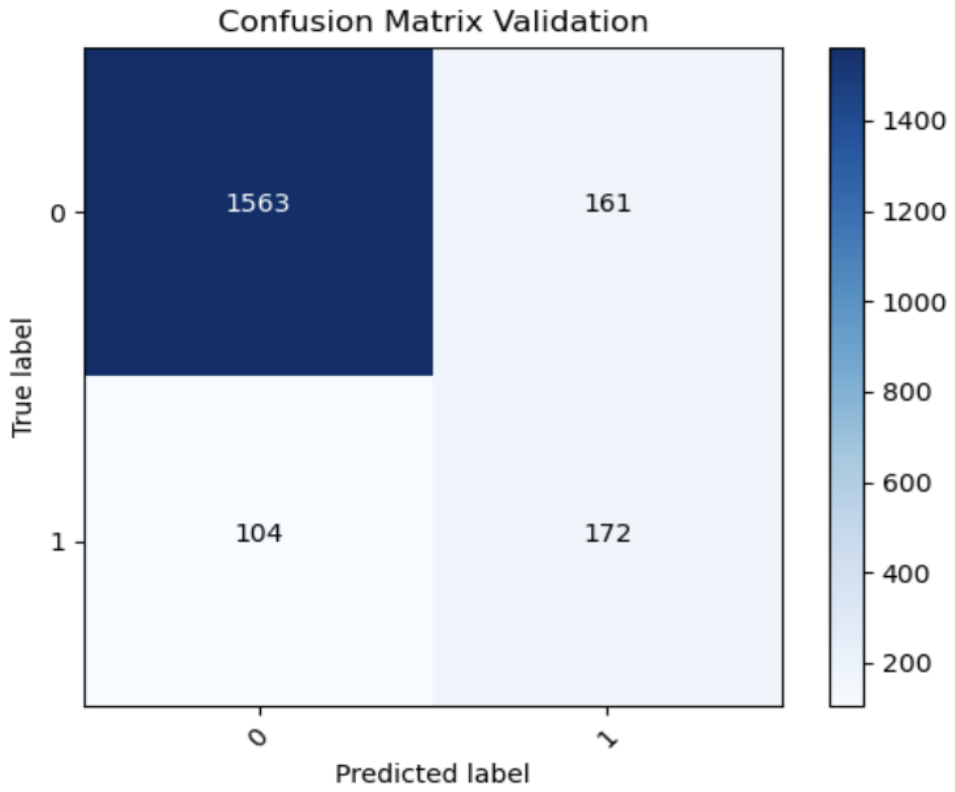
Preprocesarea datelor

Pentru preprocesarea datelor am ales libraria Pillow din Python, unde am folosit doar o creștere a contrastului. Aceasta augmentare mi-a adus un plus de 0.05 în medie algoritmului. Printre alte variante încercate s-au numărat cresterea luminozitatii imaginii și adăugarea de noise, însă nu au adus imbunatatiri.

Pentru acest algoritm am ales metoda undersampling, folosind doar 6000 de imagini din clasa 0 și pe toate din clasa 1, știind din prima vizualizare a datelor ca acestea sunt unbalanced.

Hyperparameter Tuning

După încercări repetate de a modifica hyper parametrii nu am reușit sa obțin o îmbunătățire a scorului, așadar în final am rămas cu valorile default din constructorul modelului din sklearn.
Folosind Random Forest am obținut un f1_score de 0.56, deși fiind un algoritm random și de fiecare data obțin un rezultat diferit, acesta nu iese din intervalul 0.55-0.57.



	precision	recall	f1-score	support
0.0	0.94	0.91	0.92	1724
1.0	0.52	0.63	0.57	276
accuracy			0.87	2000
macro avg	0.73	0.77	0.75	2000
weighted avg	0.88	0.87	0.87	2000

F1_SCORE ~ 0.56

Al doilea model testat - Convolutional Neural Network

Un model Convolutional Neural Network este un tip de rețea neuronala care este utilizat pentru probleme de clasificare și recunoaștere a imaginilor.
Acestea fiind spuse, am acordat cel mai mult timp și resurse acestei abordări datorită eficientei cunoscute a unui astfel de model pentru probleme similare cu cea pe care încerc sa o rezolv.

Preprocesarea datelor

Inițial am început cu metoda de undersampling folosită anterior. În urma mai multor încercări am observat că modelul face overfitting, inițial am modificat arhitectura modelului ca în final să-mi dau seama că erau prea puține date de antrenare, astfel am abandonat ideea de undersampling.

În prima instanța am încercat o metoda de oversampling, respectiv SMOTE.
SMOTE (Synthetic Minority Over-sampling Technique) consta in crearea de exemple sintetice din clasa minoritara prin interpolarea intre exemplele existente din aceasta clasa. Aceste exemple noi sunt create prin alegerea aleatorie a doua exemple din clasa minoritara si prin adaugarea unui factor de scalare intre ele.
Pentru ceva timp aceasta abordare a fost cea mai buna insa tot nu obtineam rezultatele dorite, f1 score-ul fiind în medie 0.59 cu diferite arhitecturi ale modelului.

În final am abandonat aceasta idee, si am rămas cu varianta de class weights. Class weights este un parametru ce poate fi pasat metodei de fit care ii spune modelului sa dea importanță mai mare la învățare uneia sau mai multor clase. Am calculat aceste weights folosind funcția class_weight din sklearn.utils astfel obtinand acest dictionar:
{0: 0.5876821814762576, 1: 3.351206434316354}

Desigur nu m-am bazat pe aceste calcule și am modificat puțin cate puțin weight-urile în speranta unui rezultat mai bun, însă varianta calculata exact s-a dovedit ca fiind cea mai buna.

Încercări

În implementarea CNN-ului am început de la mic, la mare atat in lățime cat și adancime.
Am ales 2 incrementari pe care vreau sa le detaliez.

În toate aceste încercări am folosit următoarele layere:
BatchNormalization - normalizeaza datele de intrare pentru a asigura că au o medie aproape de zero și o deviatie aproape de 1. Acest lucru ajută la reducerea problemelor cauzate de variația datelor de intrare și accelereaza procesul de învățare.
Conv2D - aplica un filtru (kernel) de o dimensiune data peste imagini și calculează o sumă ponderată a valorilor pixelilor din imaginea de intrare, ponderate în funcție de valorile din filtru, aceste valori ale filtrului sunt valorile invatate pe parcursul antrenarii
MaxPooling2D - parcurgem imaginile cu o matrice de dimensiune data, și se ia pixelul cu valoarea cea mai mare din acei pixeli aflați în cadrul dimensiunii matricilor, ajuta la reducerea datelor de intrare pentru următoarele layere și prin urmare, parametrii invatabili din rețea.
ReLu - functie non liniara de forma max(0, value)
Flatten - aplatizeaza inputul facandu-l de forma unui vector unidimensional
Dense - strat dens fully connected de neuroni

Alte layere încercate care nu au adus îmbunătățiri și nu apar în aceste variante:
GlobalAveragePooling2D - transforma o matrice de caracteristici într-un vector de caracteristici de dimensiune fixă
LeakyRelu - functie non liniara de forma f(x) = {x, x >= 0; alpha * x, x < 0}

Callbacks

Pe parcursul incrementarilor am folosit de fiecare data metoda ReduceLROnPlateau din keras.callbacks. Astfel după un număr dat de epoci în care loss-ul nu scade, modific learning rate-ul cu jumătatea valorii sale curente, astfel evitând zonele de convergenta lentă în antrenare.

Alte callback-uri folosite pe parcursul modelelor sunt tensorboard, early stopping si model checkpoint.

Tensorboard este un callback ce imi oferă posibilitatea de a urmări în timp real ploturile modelului în timpul învățării.
Early stopping este un callback ce imi opreste invatarea dupa un numar dat de epoci în care nu vede vreo imbunatatire în validation, astfel evitand overfitting-ul, și în plus îmi restaurează valorile weight-urilor din cea mai buna epoca.

Model checkpoint salvează fiecare epoca intr-un director dat. Acest lucru m-a ajutat în reluarea antrenarii unor epoci bune, și îmbunătățirea scorului pe acestea, și de asemenea, m-a ajutat pentru a salva și testa diferitele arhitecturi încercate.

Metrics

Pentru urmarirea eficienta a evolutiei învățării, am creat o funcție ce calculeaza f1_score-ul la finalul fiecărei epoci pentru validare, și la finalul fiecărui step pentru antrenare. Acest metric ajuta și la vizualizarea eficientă a ploturilor în tensorboard.

Augmentarea datelor

Pentru augmentarea datelor am folosit parametrii din ImageDataGenerator din keras, aceștia au fost aleși experimental și în urma unei analize manuală a datelor.
Precum la random forest, am încercat creșterea contrastului, însă în acest caz nu a adus imbunatatiri.

```
ImageDataGenerator(
    rescale=1. / 255.0,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1,
    rotation_range=15,
    fill_mode='nearest',
    horizontal_flip=True
)
```

Alți hiper parametri

Am încercat în mai multe incrementari de cnn următorii optimizatori: Adam, Adadelta, Adagrad. În repetate randuri Adam a avut rezultatele cele mai bune și prin urmare este prezent în variantele finale.
Funcția de loss aleasă este binary_crossentropy, fiind perfectă pentru aceasta problema de tip clasificare binara. Pe parcurs am incercat si o alta functie, binary_focal_crossentropy, însă fără succes.

Varianta 1

Pentru aceasta varianta au fost încercate următoarele valori:
batch_size = {20, 32, 64, 128}

image_size = {70, 100, 128, 224}
learning_rate = {0.01, 0.001, 0.005, 0.0075}
dropout = {0.2, 0.3, 0.4, 0.5}

În urma încercărilor cea mai buna varianta de batch_size s-a dovedit a fi 32, însă cu foarte mici diferente de performanta.
Dimensiunea imaginilor a fost importanta in alegere, variantele de 70 si 100 de pixeli păreau a pierde informație, performanța nefiind una dorită, iar varianta de 224 de pixeli performa similar cu cea de 128, dar cu o computatie mult mai lentă.
Pentru learning rate am mers cu varianta de 0.001, performanțele fiind similare și luand in considerare ca am implementat reducerea acestuia în zonele de convergenta lentă pe gradient.

În cadrul acestui model am încercat diferite dimensiuni ale filtrelor, cea mai buna performanta obtinandu-se cu filtre de 3x3 și 5x5 și diferite valori de stride {1,2}. Max Pooling-ul a oferit cea mai buna performanta cu dimensiunile 2x2 și strides=2.
Dropout-ul l-am ales cu valoarea de 0.2, deoarece modelul nu prezenta semne de overfitting cu aceste valori.

F1_SCORE ~ 0.65

O concluzie trasa in urma acestei iteratii a fost ca trebuie sa cresc complexitatea modelului, învățarea stagnand în aceasta varianta.

```
Conv2D(filters=32, kernel_size=3, activation='relu',
input_shape=(128, 128, 1))
BatchNormalization()
Conv2D(filters=32, kernel_size=3, activation='relu')
BatchNormalization()
Dropout(0.2)
Conv2D(filters=32, kernel_size=5, padding='same',
activation='relu', strides=2)
BatchNormalization()
MaxPooling2D(pool_size=2)
BatchNormalization()
Dropout(0.2)

Conv2D(filters=64, kernel_size=3, activation='relu')
BatchNormalization()
Conv2D(filters=64, kernel_size=3, activation='relu')
BatchNormalization()
Dropout(0.2)
Conv2D(filters=64, kernel_size=5, padding='same',
activation='relu', strides=2)
BatchNormalization()
MaxPooling2D(pool_size=2)
BatchNormalization()
```

```
Dropout(0.2)

Flatten()
Dense(64, activation='relu')
BatchNormalization()
Dropout(0.2)

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

Varianta 2

Aceasta a doua încercare s-a dovedit a fi și cea mai buna și este modelul cu scorul cel mai mare în competiție.

Pentru aceasta varianta au fost încercate următoarele valori:
batch_size = {20, 32, 64, 128}
image_size = {128, 224}
learning_rate = {0.001, 0.005, 0.0075}
dropout = {0.2, 0.3, 0.4, 0.5}

Batch size-ul cel mai bun s-a dovedit a fii din nou 32 din cele încercate.
Dimensiunea imaginii finale a fost de 224, deși viteza de computatie și parametrii antrenabili au crescut considerabil, aceasta varianta a ajuns la cel mai mare scor.
Learning rate-ul a fost ales din nou 0.001, din nou mentionand faptul ca metoda de reducere a learning rate-ului pe parcurs îmi oferă o flexibilitate în aceasta alegere, atata timp cat learning rate-ul nu este prea mare caz în care modelul ar da overshoot la weights.
În privința dropout-ului, valoarea de 0.2 a obținut cel mai bun scor, deși în finalul antrenarii early stopping-ul mi-a oprit algoritmul din cauză că începuse sa faca overfitting, creșterea dropout-ului nu i-a mai permis modelului sa ajungă la scoruri la fel de mari.

În cadrul acestui model am folosit doar filtre de dimensiune 3x3 și maxpool de dimensiune 2x2 stride 2. Am crescut complexitatea modelului adaugand atat layere in adancime cat si mai mulți neuroni/filtre în layere.

In privința overfittingului prezent la final, am încercat diferite abordări pentru a scapa de acesta. Varianta unui dropout mai ridicat nu a funcționat cum am specificat anterior. Augmentarea suplimentară datelor nu a adus îmbunătățiri în aceasta privinta. În final am încercat sa scad treptat cate un layer gandindu-ma ca, complexitatea modelului îl face sa memoreze detalii din datele de antrenare. Problema overfittingului de la final disparuse, însă performanțele atinse erau mai slabe, modelul stagnand în zona a 0.70 scor.

Astfel în final am rămas cu aceasta varianta, iar early stopping-ul mă ajuta sa opresc învățarea in cazul unui overfit, si imi restaurează weight-urile din cea mai buna epoca.

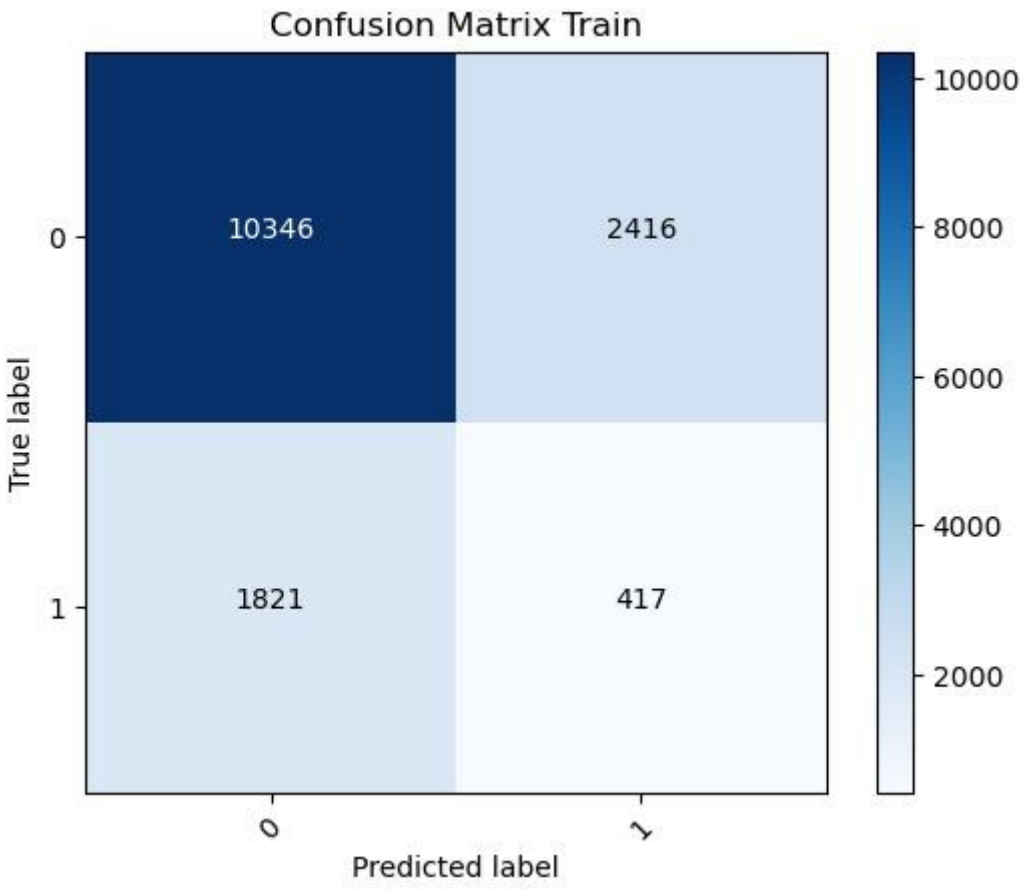
F1_SCORE ~ 0.73

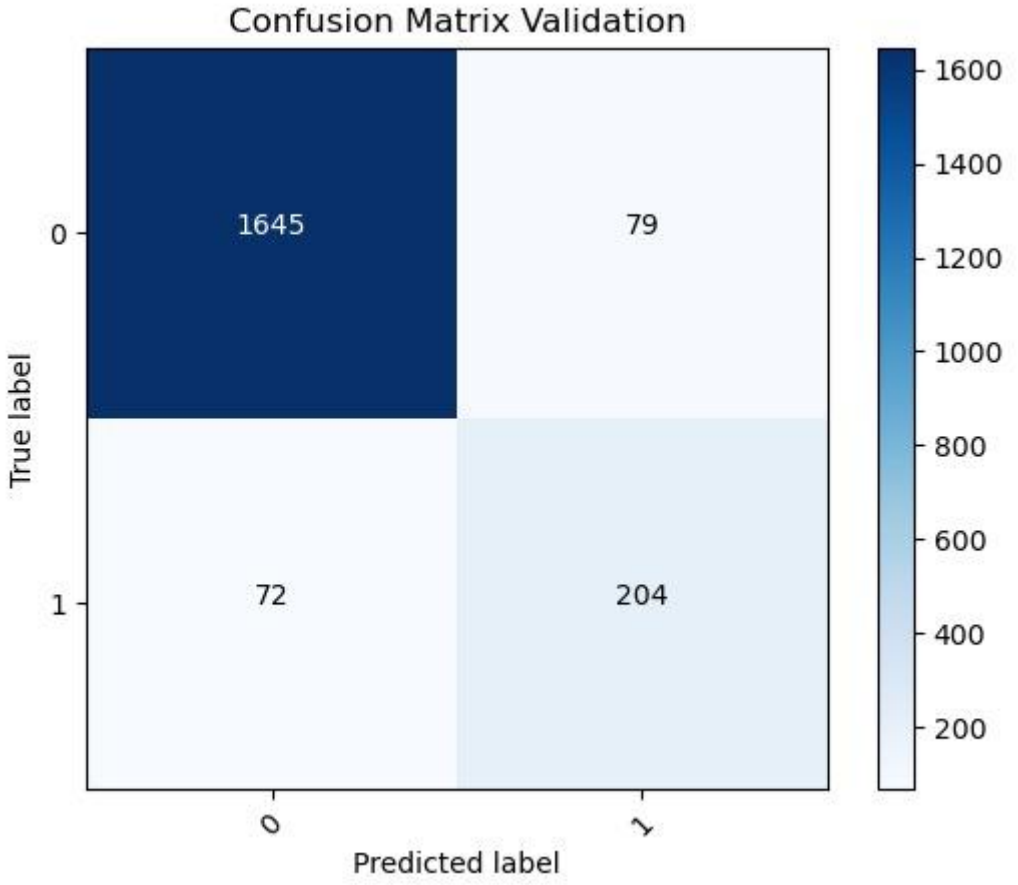
```
dropout, kernel_size = 0.2, 3
```

```
Conv2D(filters=32, kernel_size=kernel_size, padding='same',
      input_shape=(224, 224, 1)),
BatchNormalization(),
ReLU(),
Conv2D(filters=32, kernel_size=kernel_size, padding='same'),
BatchNormalization(),
ReLU(),
MaxPooling2D(),
Dropout(dropout),

Conv2D(filters=64, kernel_size=kernel_size, padding='same'),
BatchNormalization(),
ReLU(),
Conv2D(filters=64, kernel_size=kernel_size, padding='same'),
BatchNormalization(),
ReLU(),
MaxPooling2D(),
Dropout(dropout),

Conv2D(filters=128, kernel_size=kernel_size, padding='same'),
BatchNormalization(),
ReLU(),
Conv2D(filters=128, kernel_size=kernel_size, padding='same'),
BatchNormalization(),
ReLU(),
Conv2D(filters=128, kernel_size=kernel_size, padding='same'),
BatchNormalization(),
ReLU(),
MaxPooling2D(),
Dropout(dropout),

Conv2D(filters=256, kernel_size=kernel_size, padding='same'),
BatchNormalization(),
ReLU(),
Conv2D(filters=256, kernel_size=kernel_size, padding='same'),
BatchNormalization(),
```

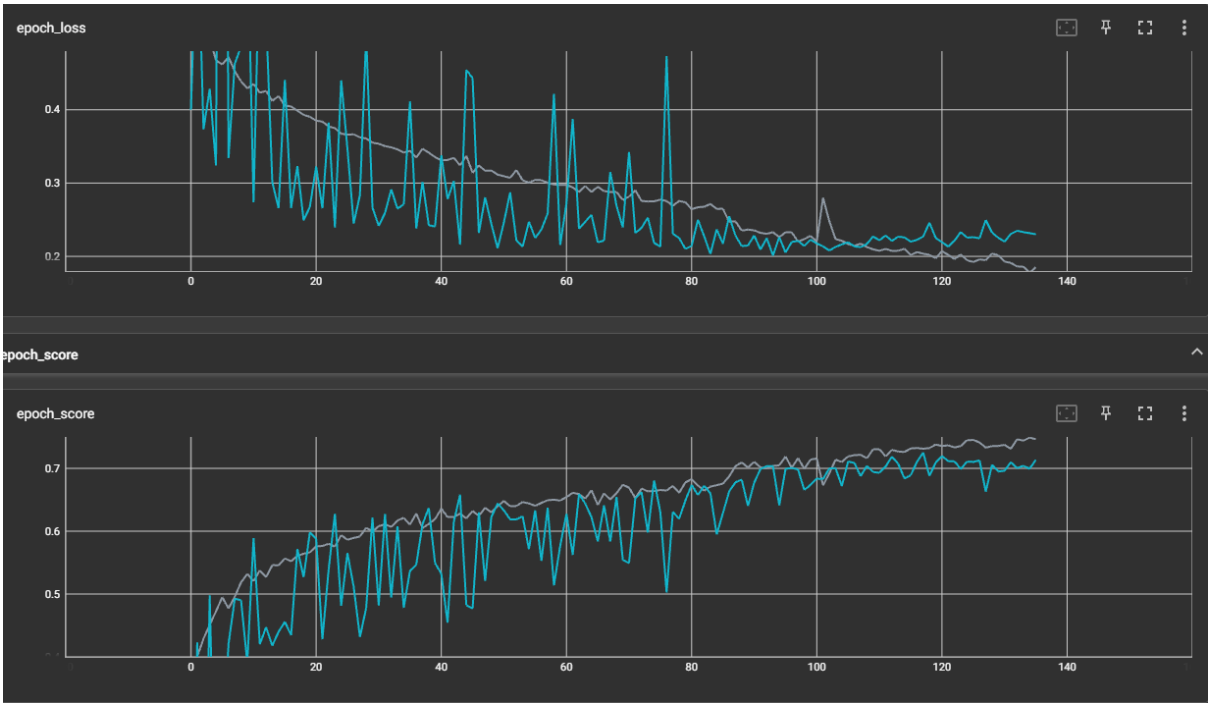
```
ReLU(),
Conv2D(filters=256, kernel_size=kernel_size, padding='same'),
BatchNormalization(),
ReLU(),
MaxPooling2D(),
Dropout(dropout),

Flatten(),
Dense(256),
ReLU(),
BatchNormalization(),
Dense(256),
ReLU(),
BatchNormalization(),
Dense(1, activation='sigmoid')
```

	precision	recall	f1-score	support
0	0.96	0.95	0.96	1724
1	0.72	0.74	0.73	276
accuracy			0.92	2000
macro avg	0.84	0.85	0.84	2000
weighted avg	0.93	0.92	0.92	2000


Confusion Matrix Train
10346
2416
1821
417
0
1
True label
0
1
Predicted label
10000
8000
6000
4000
2000


Confusion Matrix Validation
1645
79
72
204
0
1
True label
0
1
Predicted label
1600
1400
1200
1000
800
600
400
200

Evolutia antrenarii :



Evolutia antrenarii smoothened: