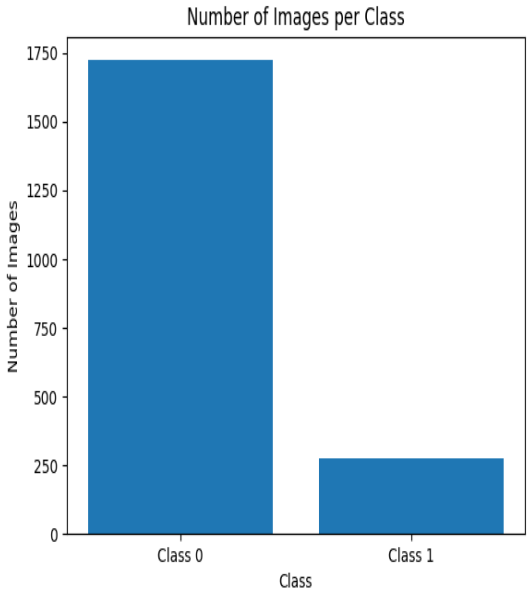
Cristea Petru-Theodor

Grupa 243

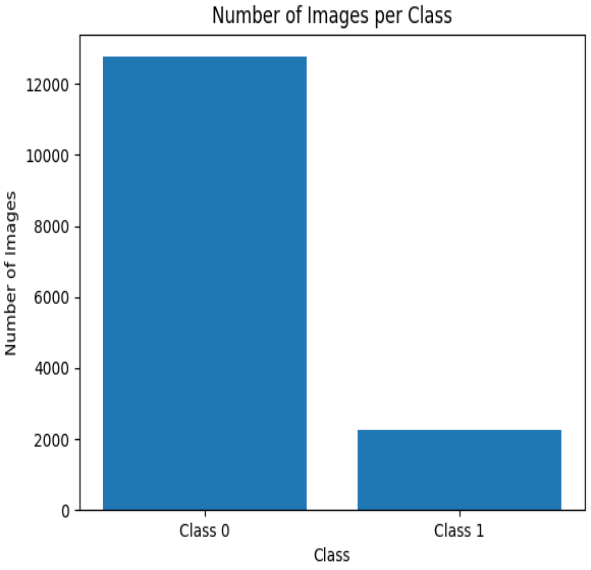
Brain anomaly detection

Proiectul constă în clasificarea unor imagini (cu dimensiuni inițiale de 224 x 224) ce reprezintă analize medicale ale creierului în două clase, și anume „0” (fără anomalii) și „1” (cu anomalie). Datele de intrare pentru acest proiect au fost 22149 de imagini, împărțite astfel:

- 15000 train data, dintre care 2238 cu label 1 și 12762 cu label 0
- 2000 validation data, dintre care 276 cu label 1 și 1724 cu label 0
- 5149 test data



Distributia labelurilor pe setul de validare



Distributia labelurilor pe setul de antrenament

Pentru a obține predicțiile pe datele de test, am implementat două modele, și anume RFC (Random Forest Classifier) și CNN (Convolutional Neural Network).

Cristea Petru-Theodor

Grupa 243

1. Random Forest Classifier

Primul model pe care l-am implementat a fost RFC. Random Forest Classifier este un model care combină mulți arbori de decizie independenți într-o "pădure" pentru a obține o predicție mai precisă și mai stabilă. Un arbore de decizie reprezintă o structură care reflectă o serie de decizii luate în funcție de input-ul furnizat, a cărei rol este de a oferi un output pe baza acestui proces. Random Forest Classifier returnează, astfel, ca predicție rezultatul mediu oferit de fiecare arbore în parte.

În comparație cu alte modele precum KNN sau SVM, acest model oferă un timp de rulare mai scurt și o acuratețe mai mare, rezultat observat în primii pași ai proiectului când am testat tot mai multe modele pentru a putea alege cel optim. De asemenea, având în vedere faptul că datele oferite ca train sunt inegal balansate (cu o pondere de aproximativ 1:6), modelul RFC a reușit să ofere predicții mult mai bune decât KNN.

Pentru a citi datele am utilizat librăria „cv2” (OpenCv) deoarece este bine optimizată pentru a lucra cu imagini, iar timpul de rulare este superior față de alte variante. De asemenea, această librărie mi-a pus la dispoziție mai multe funcții de preprocesare a imaginilor, funcții pe care le-am utilizat pentru a modifica dimensiunile imaginii, de a elimina noise-ul, de a modifica luminozitatea sau de a roti imaginea.

Preprocesarea datelor, pentru modelul RFC, s-a rezumat doar la a le redimensiona la 55% din dimensiunile inițiale și de a le aduna 10 valorilor pixelilor inițiali pentru a o face mai luminoasă, fiecare imagine transformând-o ulterior într-un np-array. Am utilizat de asemenea tehnica undersampling pentru a balansa clasele. Am încercat cu doar 6000 de imagini cu label 0, 5500 de imagini cu label 0, însă rezultatul optim a fost să iau doar 4500 de imagini cu label 0, iar pe cele cu label 1 să le utilizez integral.

În privința hiperparametrilor, am ales ca numărul de estimatori folosit să fie 700, iar adâncimea maximă să fie de 70 unități, cu ajutorul acestora obținând o valoare a F1-Score-ului de 0.58520 pe Kaggle.

Hiperparametrii testati:

Hiperparametri		**F1-score pe Kaggle**
Estimatori	**Adâncime maximă**	
100	10	0.55305
150	15	0.56187
350	25	0.57049
70	7	0.55000
700	70	0.58520

Cristea Petru-Theodor

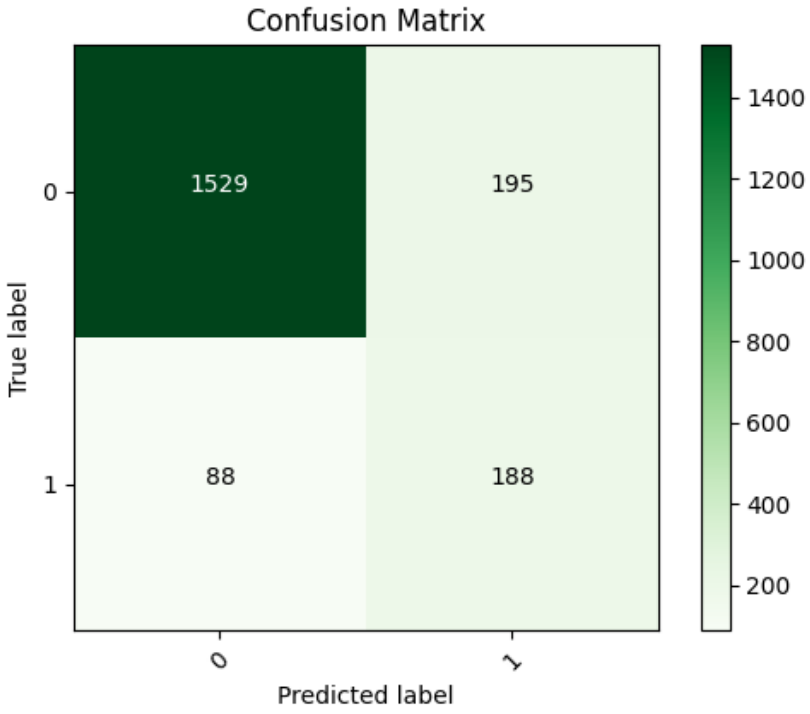
Grupa 243

Consider că acest scor ar fi putut fi doborât în cazul în care utilizăm o tehnică mai complexă pentru augmentarea datelor și renunțam la undersampling, însă am preferat să exersez aceasta tehnica pe modelul CNN. Având în vedere faptul că acest model va oferi predicții conform arborilor de decizie, vom observa faptul că F1-score-ul obținut nu va fi unul constant, ci va fluctua cu o marjă mică, dar în principiu acesta se va apropia de 0.56-0.58.

Metricile modelului RFC

Pentru fiecare model încercat, analizăm acuratețea, matricea de confuzie, precum și F1-score. De asemenea, afișăm și numărul de imagini cu anomalii (2238) precum și cele fără anomalii (4500) pentru a urmări mai ușor cât de echilibrate sunt cele două clase. Am observat, însă, faptul că nu ar trebui să mă rezum la undersampling și să renunț la date, așa că am încercat la următorul model să augmentez datele într-un mod mai complex, folosind ImageDataGenerator.

```
2238 4500
accuracy 0.8615
Confusion matrix:
 [[1536  188]
 [  89  187]]
f1 0.5745007680491552
precision              precision    recall  f1-score   support

           0       0.95      0.89      0.92      1724
           1       0.50      0.68      0.57       276

    accuracy                           0.86      2000
   macro avg       0.72      0.78      0.75      2000
weighted avg       0.88      0.86      0.87      2000
```



matricea de confuzie pentru modelul RFC

Cristea Petru-Theodor

Grupa 243

2. Convolutional Neural Network

Cel de-al doilea model pe care l-am implementat a fost CNN. Convolutional Neural Network reprezintă un model conceput pentru a putea lucra cu date de genul imaginilor. În mare, CNN-ul utilizează convoluția (care este un proces menit să extragă anumite feature-uri din imagini) și recunoaște anumite tipare pentru a încerca să ofere o predicție cât mai apropiată de adevăr. Pentru implementarea acestei rețele am utilizat Keras din TensorFlow. Legat de preprocesarea datelor, la fel ca la RFC, inițial am început cu tehnici mai simple și am păstrat undersampling-ul, procedând astfel:

• Luăm în considerare toate imaginile cu label 1 și doar 6000 cu label 0 din cele aproximativ 12000.

• Fiecare imagine o stocăm după ce o centrăm și o scălam la 90 de pixeli, iar mai apoi stocăm și versiunile rotite pe orizontală și pe verticală.

• Foloseam, de asemenea, și StandardScaler pentru a normaliza aceste imagini. Cu ajutorul acestei preprocesări am reușit să ating un F1-score de aproximativ 0.66666 pe Kaggle, însă am mai încercat încă o tehnică de augmentare a datelor, și anume folosind clasa ImageDataGenerator din librăria Keras.

Am împărțit astfel imaginile în batch-uri de 96, iar de această dată am stocat imaginile cu dimensiunea de 128x128 de pixeli. Totodată, am setat ca hiperparametru pentru funcția „flow_from_directory" class_mode="binary" (pentru o clasificare binară a imaginilor oferite ca train), iar color_mode="grayscale", deoarece când citeam datele folosind OpenCV le salvam ca „grayscale".

Legat de modificarile aduse imaginilor, foloseam urmatoarele:

- horizontal_flip = True
- rotation_range=20
- zoom_range=0.2
- shear_range=0.2
- rescale=1./255
- width_shift_range=0.15
- height_shift_range=0.15

Am testat de asemenea samplewise_std_normalization si samplewise_center, insa rezultatul maxim a fost obtinut fara acesti parametri.

Pentru a ajunge la cea mai bună versiune a arhitecturii modelului implementat, am testat mai multe versiuni cuprinzând mai multe layere, precum:

Cristea Petru-Theodor

Grupa 243

- Conv2D reprezintă un layer cu rolul de a forma caracteristici (feature-uri) specifice imaginilor de input. Funcția de activare utilizată a fost „relu” (f(x) = max(0, x)).
- MaxPooling2D, cu pool_size=(2, 2) care reprezintă layerul cu rolul de a reduce din dimensiunile imaginii luând valorile maxime obținute în urma procesului de glisare peste imaginea inițială a ferestrei de dimensiuni pool_size.
- Dropout ce reprezintă layerul care a transforma anumiți neuroni în neuroni inactivi cu o anumită probabilitate. Este esențial pentru a reduce overfitting-ul.
- Flatten care reprezintă layerul ce aplatizează datele de input.
- Dense ce reprezintă layerul ce conectează toți neuronii.

De menționat este faptul că am încercat și layerul BatchNormalization, însă am observat o evoluție mai bună a modelului meu dacă nu foloseam acest layer, iar în consecință l-am eliminat.

Din punct de vedere al hiperparametrilor, am ajuns la concluzia că pentru fiecare layer, în urma a mai multor teste, să stabilesc următoarele convenții:

• Pentru layerele de tip Conv2D am ales ca funcție de activare „relu”, kernel_size=(3, 3) și numărul de filtre să fie din mulțimea {64, 128, 256}, într-un final observând că rezultatul optim îl obțin în momentul în care utilizez doar 64 și 128. De asemenea, input_shape am setat să fie (128, 128, 1) primele două valori reprezentând înălțimea și lățimea imaginii, iar cel de-al treilea numărul canalelor.

• Pentru layerele de tip MaxPooling2D am ales ca parametrul pool_size să fie egal cu 2,2

• Pentru layerele de tip Dropout, înainte de aplatizarea inputului le setăm la 0.25 (valoare optimă aleasă în urma testelor dintre 0.2, 0.25, 0.3, 0.4), iar după aplatizare le setăm 0.5.

• Pentru layerele de tip Dense, în general, le setăm un număr mai mare de filtre, precum 192, 256, 512, iar ultimul layer de tip Dense avea un singur filtru deoarece clasam imaginile în doar două categorii, iar funcția de activare era sigmoid deoarece îmi reducea inputul la un număr din intervalul [0,1], pe care după ce îl aproximam aflam labelul prezis de model.

Pentru compilarea modelului, am folosit funcția de loss binary_crossentropy, optimizatorul Adam, pentru care am încercat și o rată de învățare dinamică (mai mare la început urmând să scadă la un anumit număr de epoci), iar pentru metrica modelului am folosit acuratețea, dar și „tfa.metrics.F1Score(num_classes=2, average='micro', threshold=0.5)” (add-on care mă ajută să printez F1-Score pentru a putea urmări mai ușor evoluția modelului). Urmarind evoluția detaliată a modelului, am apelat la o tehnică prin care salvăm epoca cu cea mai bună acuratețe, pentru a o analiza mai ușor, folosind ModelCheckPoint din keras.Callbacks.

Cristea Petru-Theodor

Grupa 243

CNN_varianta_1

Prima varianta de CNN implementata
Conv2D (32, kernel_size=(3, 3), activation="relu", input_shape=(128, 128, 1)))
Conv2D(32, kernel_size=(3, 3), activation="relu"))
BatchNormalization())
MaxPooling2D(pool_size=(2, 2)))
Dropout(0.25))
Conv2D(64, (3, 3), activation="relu"))
BatchNormalization())
Conv2D(64, (3, 3), activation="relu"))
BatchNormalization())
MaxPooling2D(pool_size=(2, 2)))
Dropout(0.25))
Conv2D(128, (3, 3), activation="relu"))
BatchNormalization())
Conv2D(128, (3, 3), activation="relu"))
BatchNormalization())
MaxPooling2D(pool_size=(2, 2)))
Dropout(0.25))
Flatten())
Dense(256, activation="relu"))
BatchNormalization())
Dropout(0.5))
Dense(128, activation="relu"))
BatchNormalization())
Dropout(0.5))
Dense(1, activation="sigmoid"))

Această rețea a reușit să atingă un F1-score de aproximativ 0.62, ceea ce s-a dovedit a fi mai apropiată de adevăr decât RFC, dar pentru această rețea încă nu am utilizat tehnicile complexe de augmentare a imaginilor.

Cristea Petru-Theodor

Grupa 243

CNN_varianta_2

Cea de-a doua varianta a CNN-ului
Conv2D(64, kernel_size=(3, 3), activation="relu", input_shape=(128, 128, 1)))
Conv2D(64, kernel_size=(3, 3), activation="relu"))
MaxPooling2D(pool_size=(2, 2)))
Dropout(0.25))
Conv2D(128, (3, 3), activation="relu"))
Conv2D(128, (3, 3), activation="relu"))
MaxPooling2D(pool_size=(2, 2)))
Dropout(0.25))
Conv2D(128, (3, 3), activation="relu"))
Conv2D(128, (3, 3), activation="relu"))
MaxPooling2D(pool_size=(2, 2)))
Dropout(0.25))
Conv2D(256, (3, 3), activation="relu"))
Conv2D(256, (3, 3), activation="relu"))
MaxPooling2D(pool_size=(2, 2)))
Dropout(0.25))
Flatten())
Dense(512, activation="relu"))
Dropout(0.5))
Dense(256, activation="relu"))
Dropout(0.5))
Dense(1, activation='sigmoid'))

A doua abordare a acestui model a constat in modificarea numarului de filtre, fapt care a dus la imbunatatirea scorului F1 pana la aproximativ 0.66666 pe Kaggle.

Tot pentru cea de-a doua abordare, am folosit ulterior ImageDataGenerator, cu batch_size de 32, iar când dădeam „fit” modelului specificam parametrul class_weights setat cu „{0:1, 1:2}” și să ruleze pe 175 epoci. Această tactică a dus la creșterea scorului până la aproximativ 0.68 pe Kaggle datorită augmentării mult mai complexe. Totodată, pentru a evita overfitting-ul aici am folosit EarlyStopping (modelul se oprea în momentul în care „val_loss” își schimba monotonia de 10 ori), însă ulterior am renunțat la această tehnică deoarece am observat că există posibilitatea găsirii unei epoci optime pe care să o omit.

Cristea Petru-Theodor

Grupa 243

CNN_varianta_3

Cea de-a treia varianta a CNN-ului
Conv2D(64, kernel_size=(3, 3), activation="relu", input_shape=(128, 128, 1))
Conv2D(64, kernel_size=(3, 3), activation="relu")
MaxPooling2D(pool_size=(2, 2))
Dropout(0.25)
Conv2D(128, (3, 3), activation="relu")
Conv2D(128, (3, 3), activation="relu")
MaxPooling2D(pool_size=(2, 2))
Dropout(0.25)
Conv2D(128, (3, 3), activation="relu")
Conv2D(128, (3, 3), activation="relu")
MaxPooling2D(pool_size=(2, 2))
Dropout(0.25)
Flatten()
Dense(256, activation="relu")
Dropout(0.5)
Dense(1, activation='sigmoid')

Această versiune a rețelei, împreună cu augmentarea complexă a datelor, a obținut un scor de 0.7207 pe Kaggle. Modelul a fost instruit pe 400 de epoci, cu parametrul class_weights setat la „{0:1, 1:1.5}”, de asemenea batch_size-ul din ImageDataGenerator a fost setat la 96.

Pentru aceasta versiune am incercat totodata modificarea treptata learning rate-ului, dar nu am avut o imbunatatire evidenta, fapt care m-a determinat sa renunt la aceasta tehnica.

Aceasta este functia menita sa imi modifice learning rate-ul.

De asemenea, pentru a putea lucra cu aceasta tehnica am apelat la LearningRateScheduler care primea ca parametru aceasta functie si pe care l-am adaugat in callback-urile modelului.

```
def invscaling_lr(epoch, lr):
    if epoch == 0:
        return lr
    elif epoch % 7 == 0:
        return lr / (epoch ** 0.33)
    else:
        return lr
```

Cristea Petru-Theodor

Grupa 243

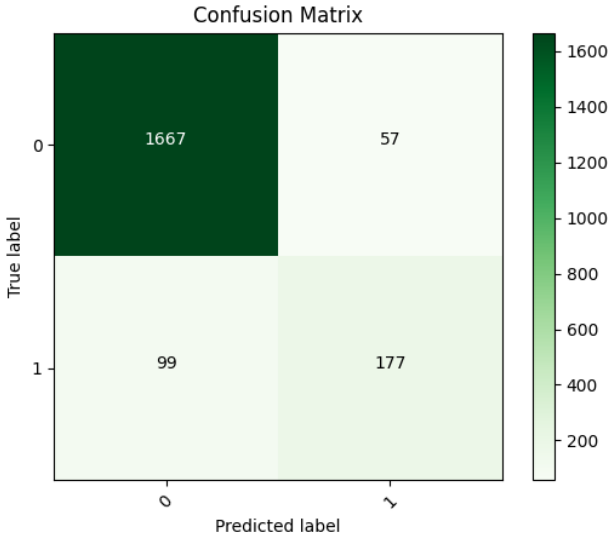
În urma antrenării pe 300 de epoci, ultimul model discutat a obținut câteva rezultate interesante care s-au dovedit a fi optime față de celelalte modele încercate. De exemplu, am remarcat următoarele:

- Epoch 186/300

157/157 [==============================] - 29s 182ms/step - loss: 0.2212 - accuracy: 0.9305 - f1_score: 0.7581 - val_loss: 0.2221 - val_accuracy: 0.9220 - val_f1_score: 0.6941

Metricile epocii 186 ale acestui model CNN

```
161/161 [==============================] - 2s 15ms/step
63/63 [==============================] - 1s 15ms/step
Accuracy  0.922
f1_score  0.6941176470588235
Confusion matrix:
 [[1667   57]
 [  99  177]]
classification report              precision    recall  f1-score   support

           0       0.94      0.97      0.96      1724
           1       0.76      0.64      0.69       276

    accuracy                           0.92      2000
   macro avg       0.85      0.80      0.82      2000
weighted avg       0.92      0.92      0.92      2000
```



Matricea de confuzie pentru epoca 186

Cristea Petru-Theodor

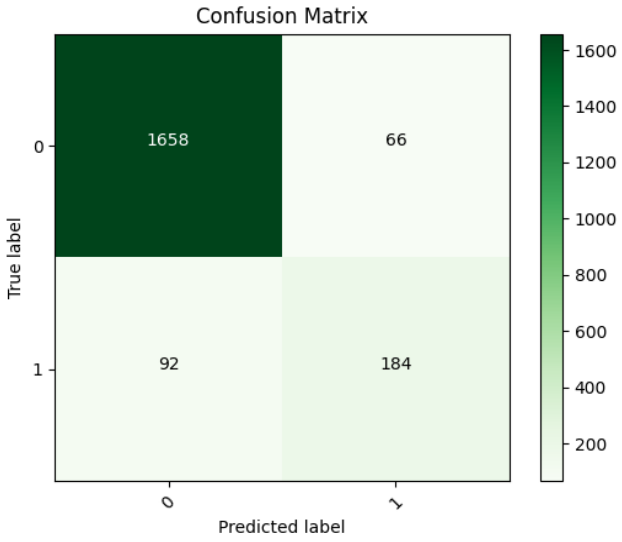
Grupa 243

- Epoch 219/300

157/157 [==============================] - 29s 181ms/step - loss: 0.2117 - accuracy: 0.9306 - f1_score: 0.7607 - val_loss: 0.2333 - val_accuracy: 0.9210 - val_f1_score: 0.6996

Metricile epocii 219 ale acestui model CNN

```
161/161 [==============================] - 2s 15ms/step
63/63 [==============================] - 1s 14ms/step
Accuracy  0.921
f1_score  0.6996197718631179
Confusion matrix:
 [[1658   66]
 [  92  184]]
classification report              precision    recall  f1-score   support

           0       0.95      0.96      0.95      1724
           1       0.74      0.67      0.70       276

    accuracy                           0.92      2000
   macro avg       0.84      0.81      0.83      2000
weighted avg       0.92      0.92      0.92      2000
```



Matricea de confuzie pentru epoca 219

Cristea Petru-Theodor

Grupa 243

Astfel, utilizând modelul CNN am reușit să ajung la un scor maxim pe Kaggle de 0.72. Comparând cele trei variante menționate anterior, se observă faptul că factori precum arhitectura modelului, la fel și augmentarea datelor au fost decisive pentru îmbunătățirea capacității modelului.

O arhitectură mult prea complexă nu ar fi fost potrivită deoarece există riscul Overfitting-ului, iar o augmentare sumară a datelor sau un UnderSampling nu ar fi ajutat la creșterea scorului. Arhitectura modelului care a obținut cel mai precis rezultat din punct de vedere al F1-score avea un total de 5.273.281 de parametri antrenabil, nefiind cea mai mare arhitectură pe care am testat-o.

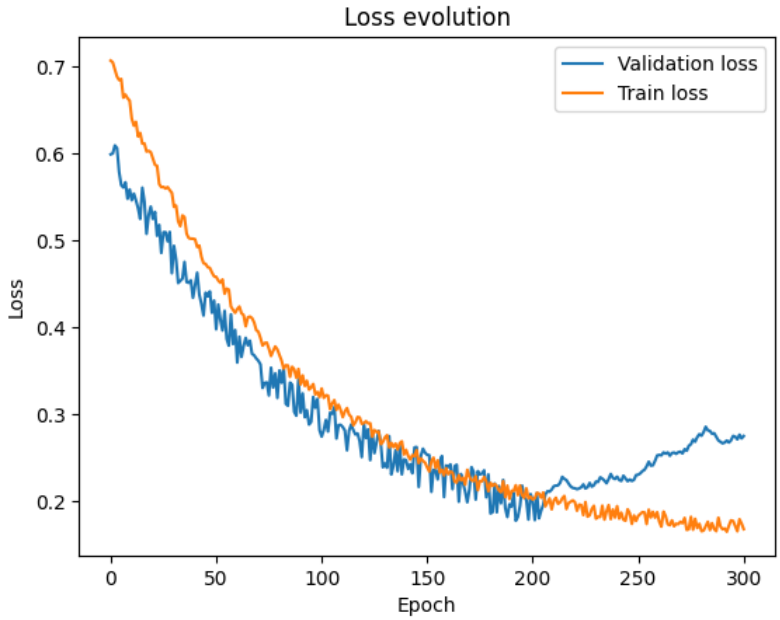
Așa arată în detaliu structura modelului utilizat, pentru fiecare layer precizând numărul de parametri precum și forma outputului..

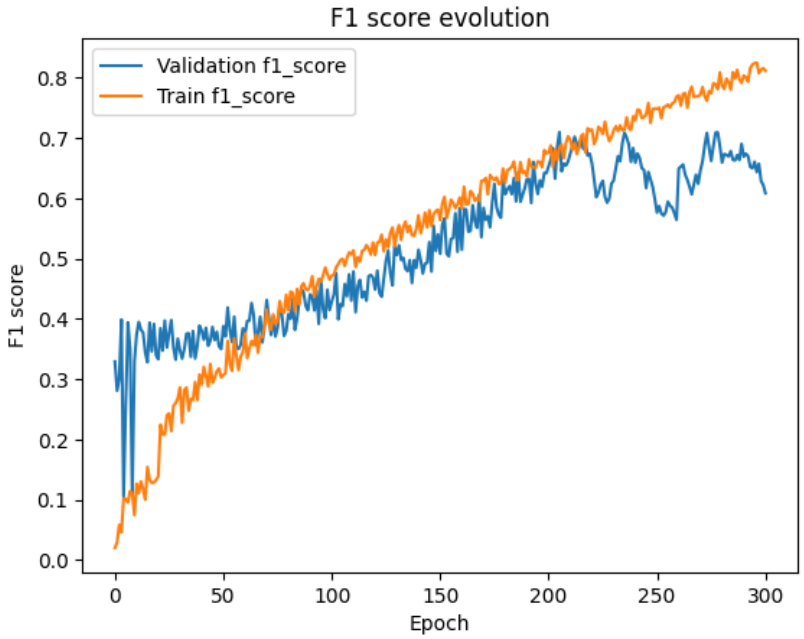
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 64)      640

 conv2d_1 (Conv2D)           (None, 124, 124, 64)      36928

 max_pooling2d (MaxPooling2D  (None, 62, 62, 64)       0
 )

 dropout (Dropout)           (None, 62, 62, 64)        0

 conv2d_2 (Conv2D)           (None, 60, 60, 128)       73856

 conv2d_3 (Conv2D)           (None, 58, 58, 128)       147584

 max_pooling2d_1 (MaxPooling  (None, 29, 29, 128)      0
 2D)

 dropout_1 (Dropout)         (None, 29, 29, 128)       0

 conv2d_4 (Conv2D)           (None, 27, 27, 128)       147584

 conv2d_5 (Conv2D)           (None, 25, 25, 128)       147584

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

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

 flatten (Flatten)           (None, 18432)             0

 dense (Dense)               (None, 256)               4718848

 dropout_3 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 1)                 257

=================================================================
Total params: 5,273,281
Trainable params: 5,273,281
Non-trainable params: 0
```

Cristea Petru-Theodor

Grupa 243

Pentru a putea urmări mai ușor evoluția acestui model și implicit pentru a putea analiza comportamentul său, am hotărât să urmăresc grafurile care redau evoluția F1-score-ului, precum și a „loss-ului”.





Cristea Petru-Theodor

Grupa 243

Scurt rezumat al variantelor de modele CNN pe care le-am testat, menționând arhitectura și tehnicile de preprocesare a datelor, precum și scorul obținut pe platforma Kaggle:

Varianta de CNN	**Tehnici**	**Scor obtinut pe Kaggle**
CNN_varianta_1	Reteaua din tabel Undersampling	0.62
CNN_varianta_2	Reteaua din tabel ImageDataGenerator (class_weights-ul setat nu a fost optim) EarlyStopping	0.66666
CNN_varianta_3	Reteaua din tabel ImageDataGenerator Learning Rate dinamic	0.72

Cristea Petru-Theodor

Grupa 243

Referinte

1. *Cursuri si laboaratoare*
2. *https://scikit-learn.org/stable/*
3. *https://keras.io/examples/*
4. *https://numpy.org/*
5. *https://www.tensorflow.org*
6. *https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator*
7. *https://www.python.org/doc/*