**Documentatie proiect – Inteligenta Artificiala**

**Pomparau Renato Emil**

**Grupa 241**

**Dataset Description**

**The task is to discriminate between two classes of brain CT scans, one that contains anomalies (label 1) and one that is normal (class 0). Each sample is a grayscale image of 224x224 pixels.**

**Each example is assigned to one of the two classes. The training set consists of 15,000 labeled examples. The validation set consists of 2,000 labeled examples. The test set consists of another 5,149 examples. The test labels are not provided with the data.**

**File descriptions**

* **data.zip - the image samples (one sample per .PNG file)**
* **train\_labels.txt - the training labels (one label per row)**
* **validation\_labels.txt - the training labels (one label per row)**

**Binary Logistic Regression Model**

## In the first stage, I decided to train a Logistic Regression Model. In order to be able to train the model using the given dataset, I had to read the data and preprocess it accordingly. After using diferent data structures to manage the pixels for each image and the label assigned to it, representing representing whether it has an abnormally or not, I converted my data into one dimension array using numpy.. I experimented with Standardization (x-mean(x)/std(x)) and also with the process of normalization by dividing each pixel by 255 in order to have values between 0 and 1. The purpose of normalization is to bring all features onto a similar scale, which can help the model learn more effectively and avoid issues with numerical stability.

## After the data was read and preprocessed, all I had to do is test different configurations of hyperparameters

## Regularization strength (C) : Smaller value prevents overfitting

## Maximum number of iterations (max\_iter): to make sure it converges, I used 2000

## Solver (solver): I achieved better result with the default.

## Penalty: Used the default l2.

## In the end, I achieved :

## Confusion matrix:

## [[1555 169]

## [ 203 73]]

## Accuracy:

## 0.814

## F1\_score according to submission\_sample:

## 0.29818

## In the second stage, I tried to improve my f1\_score for the Kaggle competition so i used a CNN. This time I used a notebook from Kaggle platform due to the fact that CNN required training on GPU and i was not able to make it work on mine.

## Reading the data was the same as above, for normalization i used the same aproaches but this time the arrays were reshaped like this. I tried some preprocessing like augumenting the data and deleting the data that i considered useless, like CT Scans that were actually blank/just black but the improvement was minimal for augumenting and even worse for deletion.

## 

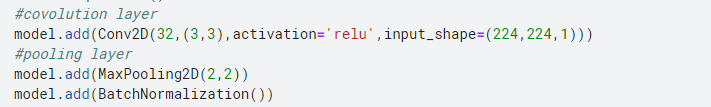
**Standardization:**

## 

**Or Normalization:**

## 

**To avoid Memory Overload I used instead a BatchNormlization Layer for each Convolutional layer**





## model = Sequential(): creates a new Sequential model object, which allows us to stack layers in sequence.

## model.add(Conv2D(32,(3,3),activation='relu',input\_shape=(224,224,1))): adds a Convolutional layer to the model with 32 filters, a kernel size of 3x3, ReLU activation function, and an input shape of (224, 224, 1). This layer will perform convolution operation on the input image to extract features.

## model.add(MaxPooling2D(2,2)): adds a MaxPooling layer to the model with a pool size of 2x2. This layer will downsample the output of the previous Convolutional layer, keeping only the most important features.

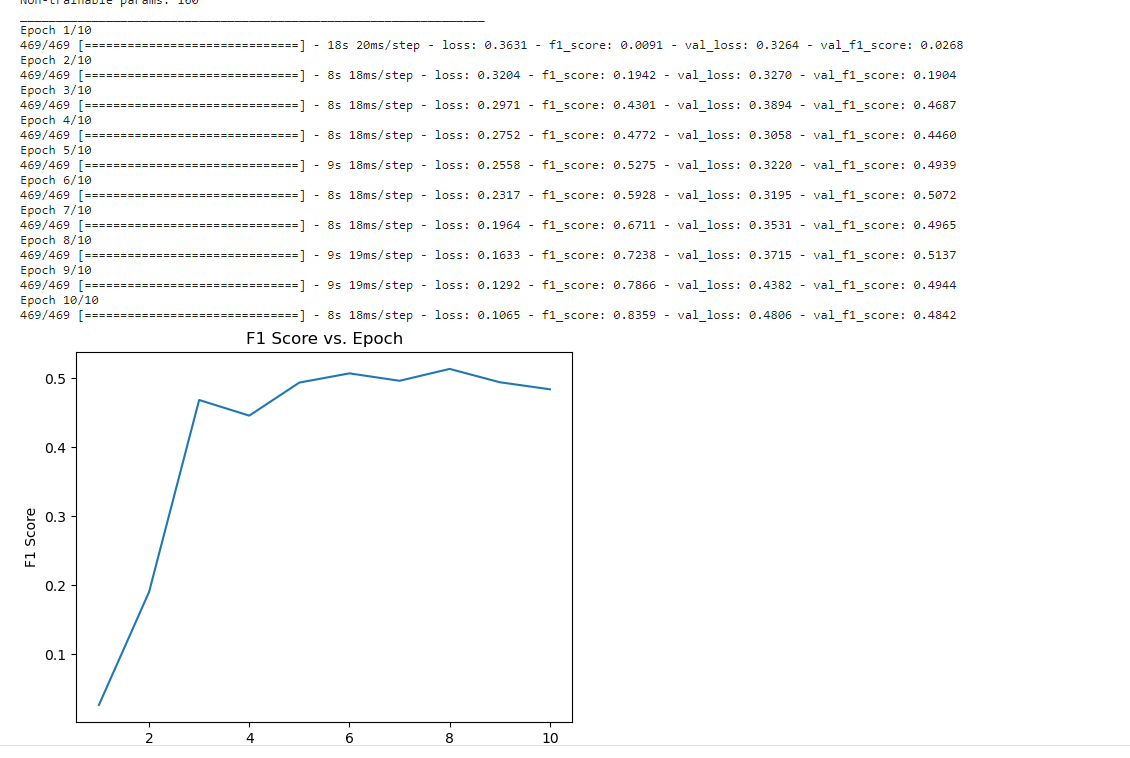
## model.add(BatchNormalization()): adds a Batch Normalization layer to the model. This layer will normalize the output of the previous layer, which helps to improve the performance of the model.

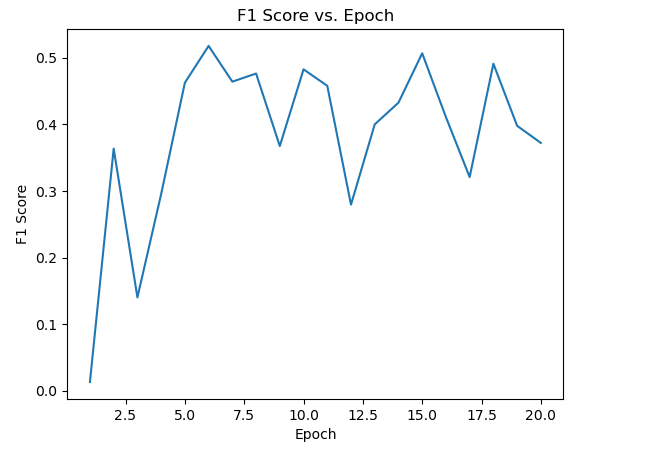
## The code repeats the above pattern of adding Convolutional, MaxPooling, and Batch Normalization layers two more times with different filter sizes (32 and 16) to further extract features from the input image and downsample the output.

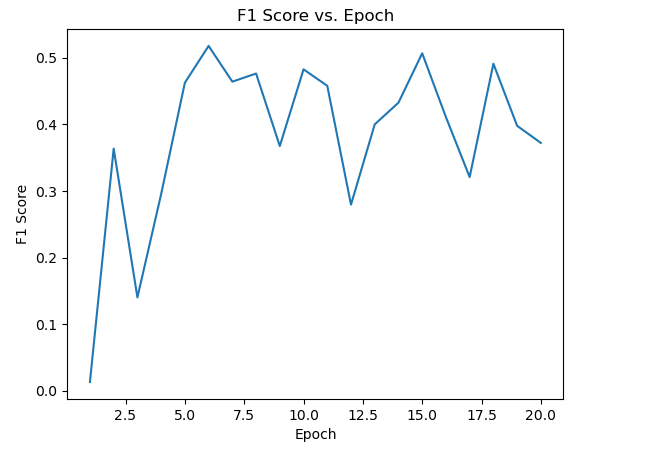


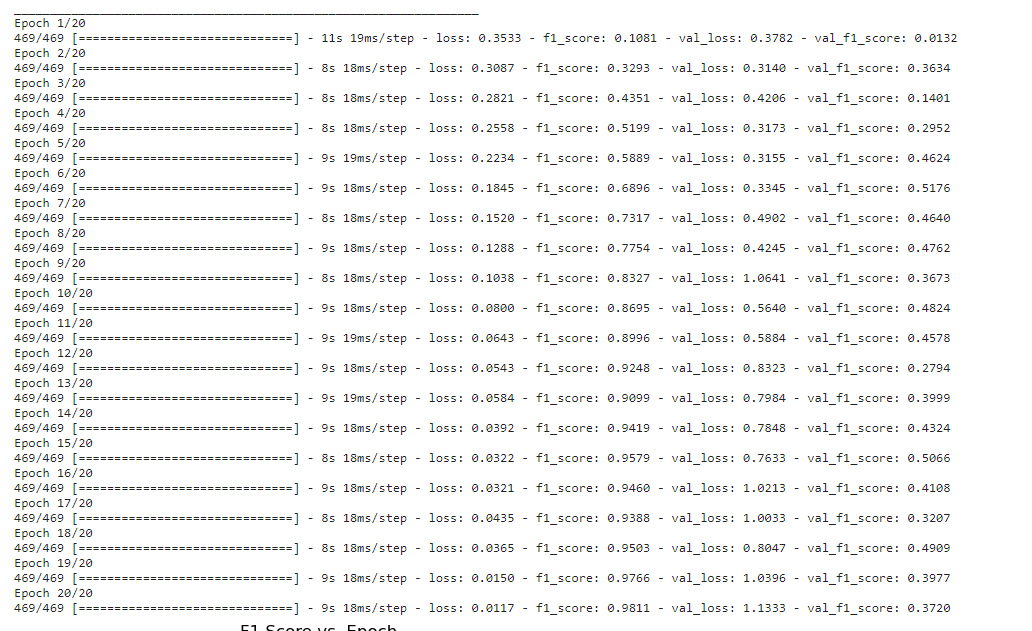
**F1\_score: 0.50413**

**For 10 epochs I achieved the highest f1\_score on submission.**









**Conclusion**

**In the end, the best solution was obtain using a CNN with 3 Convolutionals layers each followed by a MaxPooling layer and a BatchNormalization layer.**

**For the normalization, I got the highest f1\_score by dividing by each pixel by 255. However, I changed to BatchNormalization due to the Memory Overload.**

**First Convolutional Layer : (32,(3,3), activation='relu',input\_shape=(224,224,1))**

**The layer has 32 filters and the kernel dimension is 3x3.**

**By using ReLU activation function in a Conv2D layer, the negative values in the output feature maps are set to 0. This introduces non-linearity into the output and helps the network learn more complex and discriminative features.**

**Input shape was set to a matrix of 224x224 and 1 represent the grayscale configuration of the image.**

**After trial and error, I ended up with the actual configuration of the other layers.**

**The Flatten layer in a CNN is used to convert the 3D output tensor of the last convolutional layer into a 1D vector that can be fed into a fully connected (Dense) layer. The Flatten layer doesn't have any trainable parameters, it just reshapes the output tensor. After flattening the output tensor, we add a fully connected layer with 64 neurons and a ReLU activation. The ReLU activation function is commonly used in neural networks as it can introduce non-linearity into the model, making it capable of approximating complex non-linear functions. Finally, we add an output layer with a single neuron and a sigmoid activation function using. The sigmoid function is commonly used in binary classification problems as it maps the output of the model to a probability value between 0 and 1, representing the probability that the input belongs to the positive class.**

**For compiling the model, I used adam optimizer which update the weights in the model based on the error.**

**For loss function, I used Binary crossentropy, used for binary classification problems, where the output of the model is a probability value between 0 and 1. The loss function calculates the difference between the predicted probability and the actual label, and tries to minimize this difference during training.**

**In the end, my predicted\_label given by the model represent the probability to be in a certain class. If above 0.5 it means that it detected an anomaly and i changed the value to 1, else it means it detected no anomaly so the value was converted to 0.**

