Deep Learning Project

Hanciu Raluca-Maria

Gr.511

1. Dataset

The dataset contains magnetic resonance images. This is a multi-label classification task, meaning that we have to accurately classify the medical images with respect to three possible classes, where each image might belong to more than one class.

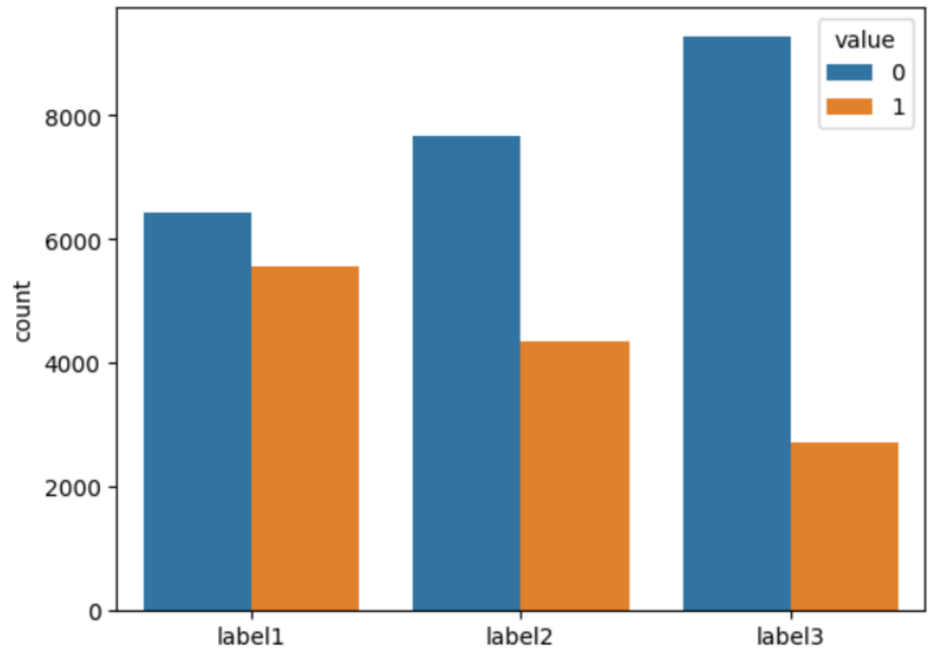
The trainset is composed of 12.000 images, the validation set of 3.000 images and the test set of 5000 images.

2. Data visualization:

All images are of the format 64,64 pixels.

Below an example of an image from the training set:

Some data vizualization of the labels values:



3. Models Used

For this classification task I have used the below 2 models:

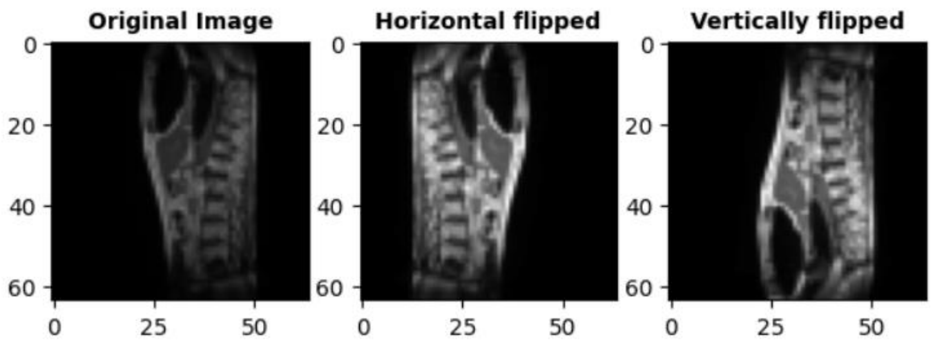
- Convolutional Neural Network
- Dense Neural Network

I. CNN

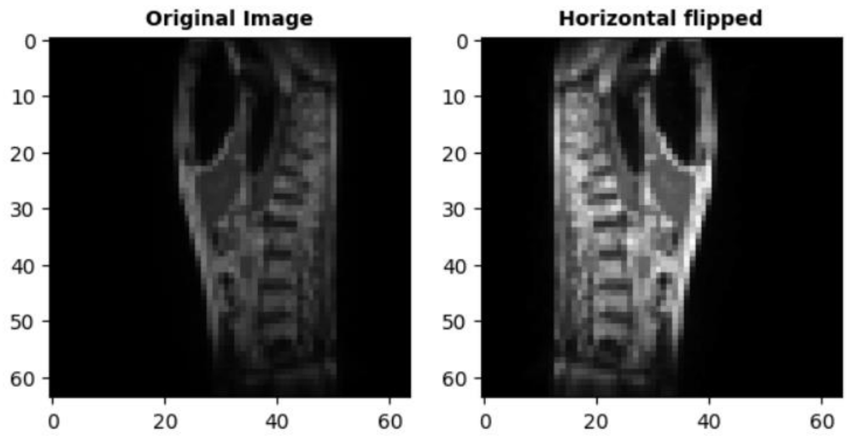
A CNN is a particular type of network design for deep learning algorithms that is useful for tasks like image recognition and pixel data processing.

For data preprocessing I have normalized the images while loading them by dividing the arrays to 255, so each pixel has a value between 0 and 1.

I have tried using data augmentation by flipping the images vertically and horizontally to obtain some more data and maybe improve the model efficiency.

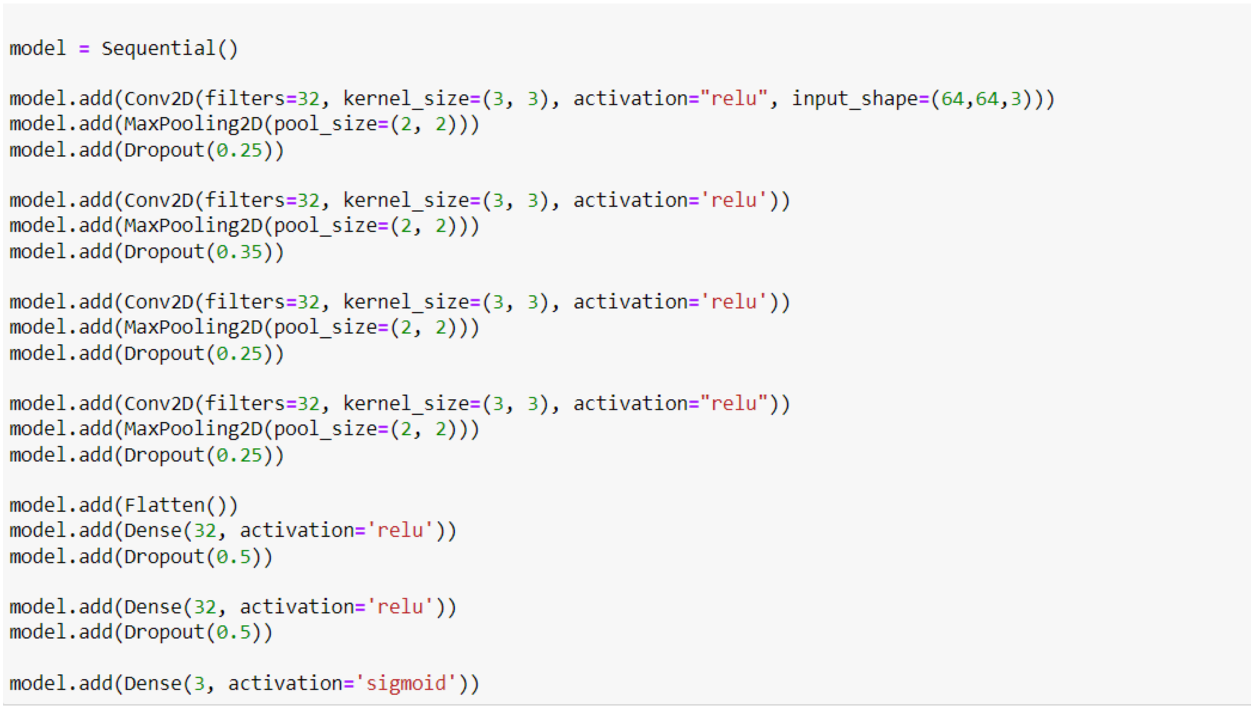


I have observed that using both augmentations my model is not improved and actually using just one augmentation (horizontally) helped my model perform better, so I decided to keep only this type of augmentation:



Also, I have observed that the data is pretty much imbalanced, meaning that there are a lot of images that have all labels = 0 (6153 out of 12000 training data have label1 = label2= label3 = 0). What I have tried to do when augmenting the data was to augment only the images that do not have all labels = 0. However, this approach has not led to any improvements so in my final models I have trained and augmented all data.

First model I have implemented is CNN, below is the architecture:



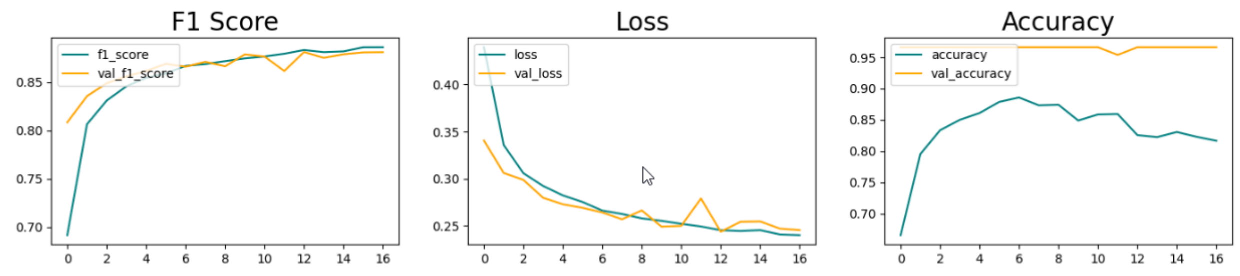
```
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3, 3), activation="relu", input_shape=(64,64,3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.35))

model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters=32, kernel_size=(3, 3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(3, activation='sigmoid'))
```

I have 4 convolution layers and 4 MaxPooling layers. Each convolution layer is followed by MaxPooling and then by a Dropout layer. I have tried many different approaches including no dropout or dropout only after the flattening layer, tried with a higher number of filters and I found that having a skinnier architecture helps me achieve better performance. I have chosen to have higher number of layers because we are dealing with medical images and then we should extract more features to improve the model. Also, the skinnier architecture performed better (along with dropout) so I could prevent overfitting.

I have used Adam optimizer with the default learning rate of 0,001. I have tried using both smaller or higher learning rate but the model was not improved. I have also tried different optimizers (SGD, RMSprop) and other loss types (categorical crossentropy, Poisson) and had unimproved results.

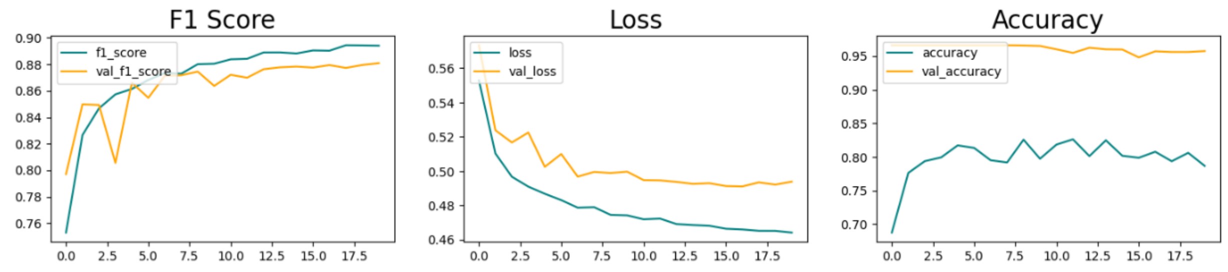
I have searched and I have seen that there is no f1 score in keras metrics, so I have used a function to track this f1 during my training process so I could plot it and visualize all metrics side by side.

Below my result with the CNN architecture:

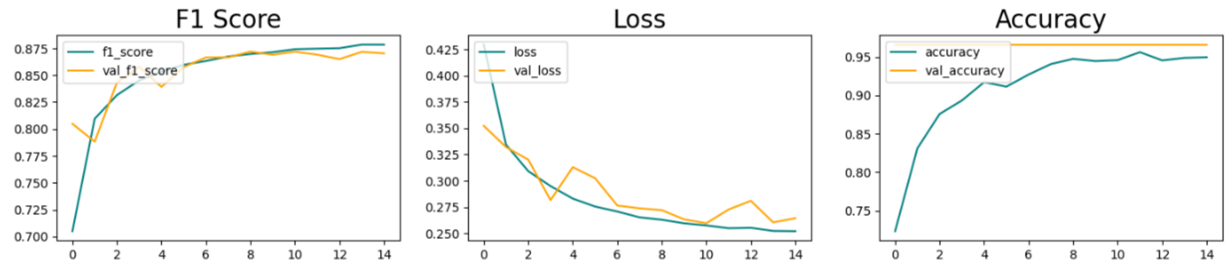


```
                precision    recall  f1-score

     label 1        0.91      0.92      0.92
     label 2        0.87      0.87      0.87
     label 3        0.85      0.81      0.83

   micro avg        0.88      0.88      0.88
   macro avg        0.88      0.87      0.87
weighted avg        0.88      0.88      0.88
 samples avg        0.43      0.44      0.42
```

Below the result with Poisson loss and Adam optimizer:



Below the result with optimizer RMSprop and binary crossentropy loss:



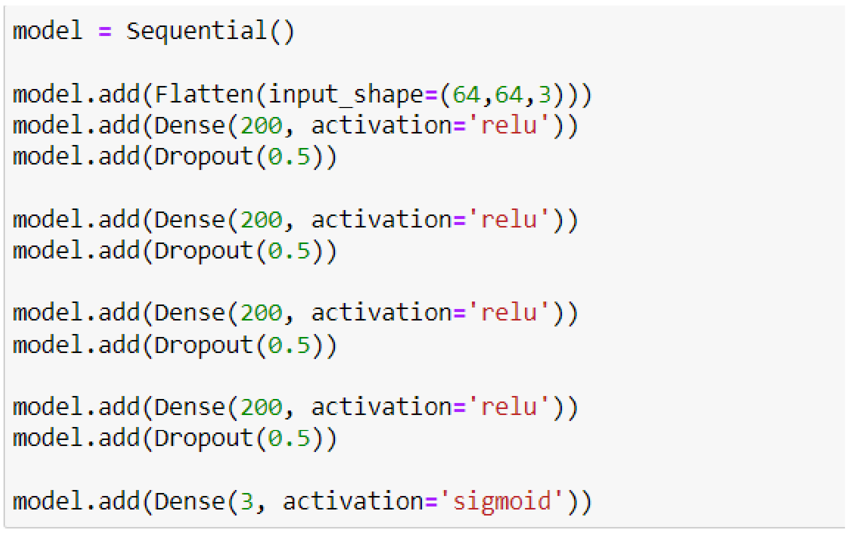
I have chosen a batch size of 64 and have tried with different sizes but looks like for my model between 50-70 is a good choice for the number of batches.

Also, I have used Early Stopping with patience=4 and weights restoring to be sure my model has the weights from the best epoch. I have initialized 20 epochs but my model uses the weights from about epoch 13-15.

II. Dense Neural Network

Dense neural networks are the simplest network architecture that can be used to fit classification models.

I have implemented the following architecture:

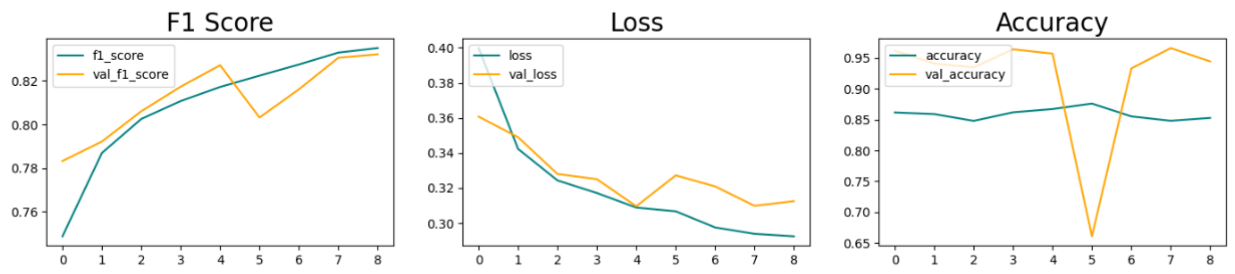


```
model = Sequential()

model.add(Flatten(input_shape=(64,64,3)))
model.add(Dense(200, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(200, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(200, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(200, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(3, activation='sigmoid'))
```

For this dense neural network, I have used a higher number of units than in CNN but also the dropout rate is higher on each layer than on CNN. Using an architecture with fewer nodes resulted in an underperforming output, whereas without dropout I had the problem of overfitting.

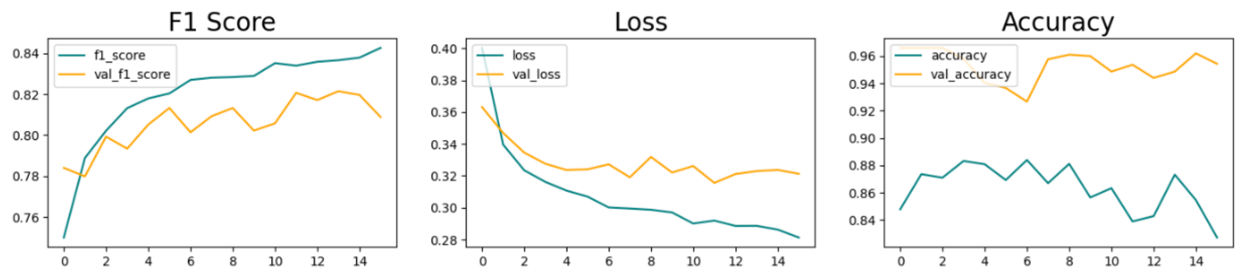
Also, for this model I have observed that I get improved performance (less overfitting) if I use both types of data augmentation – meaning horizontally and vertically flipping of the images.

I have tried at first using the same parameters for this Dense Model as I used for CNN, meaning Adam optimizer, learning rate 0.001 and binary crossentropy loss, 64 batches and I have got the following result:



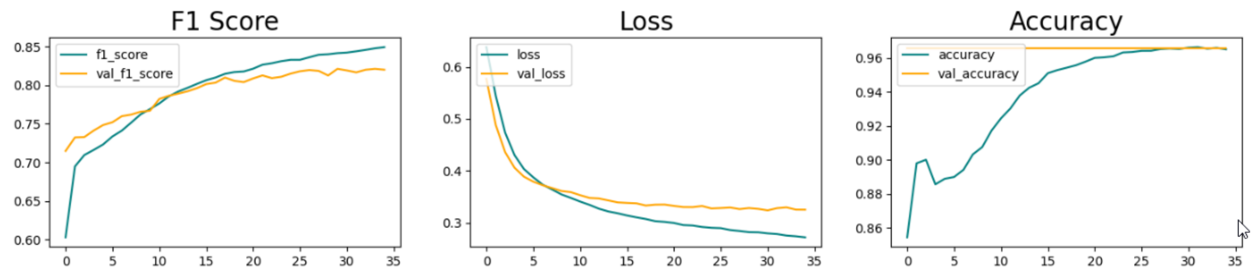
```
              precision    recall  f1-score   support

     label 1       0.87      0.94      0.90      1447
     label 2       0.81      0.87      0.84      1153
     label 3       0.75      0.52      0.61       684

   micro avg       0.83      0.83      0.83      3284
   macro avg       0.81      0.77      0.79      3284
weighted avg       0.83      0.83      0.82      3284
 samples avg       0.42      0.42      0.41      3284
```

I have also tried using a slightly higher batch = 80:

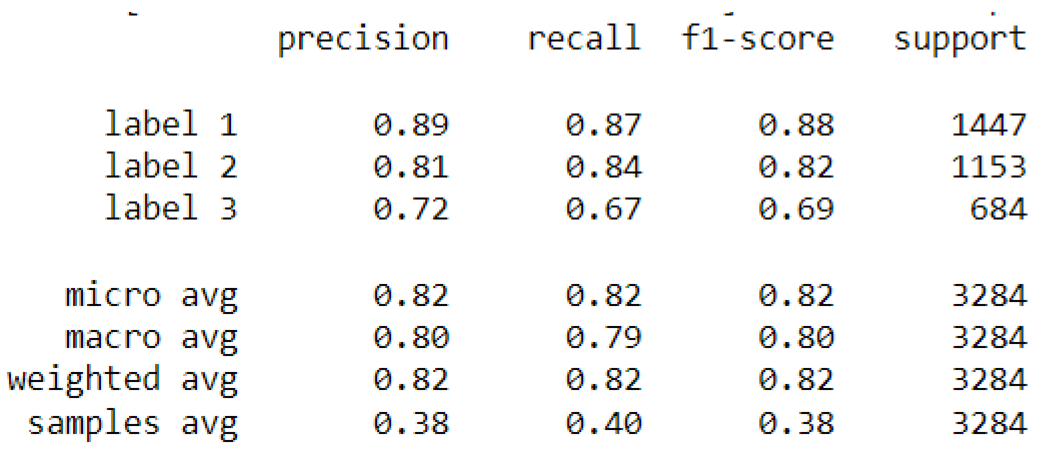


We can see that the accuracy is fluctuating and the model is not improved.

Here again I have tested with different optimizers and learning rates and I have seen that using SGD with a learning rate of 0.01 helps my model achieve better results. Also because my loss kept decreasing after 20 epochs (how I initially set the model) I have decided to increase this number to 50. Again for this dense network I have used Early Stopping and at about epoch ~30 my models stops and resets its weights to the ones from about epoch ~26.



Even though the accuracy is pretty high, we can see that with respect to f1 score the CNN model is performing much better.



```
              precision    recall  f1-score   support

     label 1       0.89      0.87      0.88      1447
     label 2       0.81      0.84      0.82      1153
     label 3       0.72      0.67      0.69       684

   micro avg       0.82      0.82      0.82      3284
   macro avg       0.80      0.79      0.80      3284
weighted avg       0.82      0.82      0.82      3284
 samples avg       0.38      0.40      0.38      3284
```