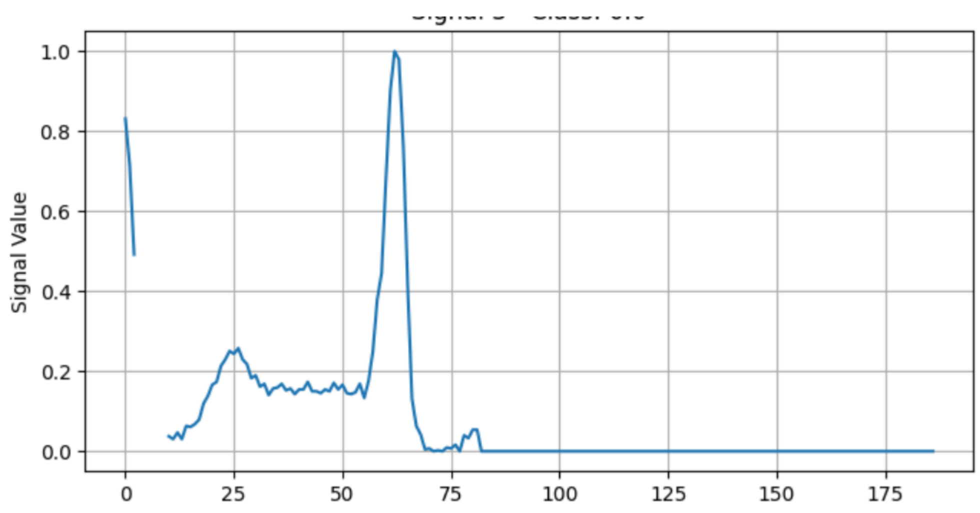
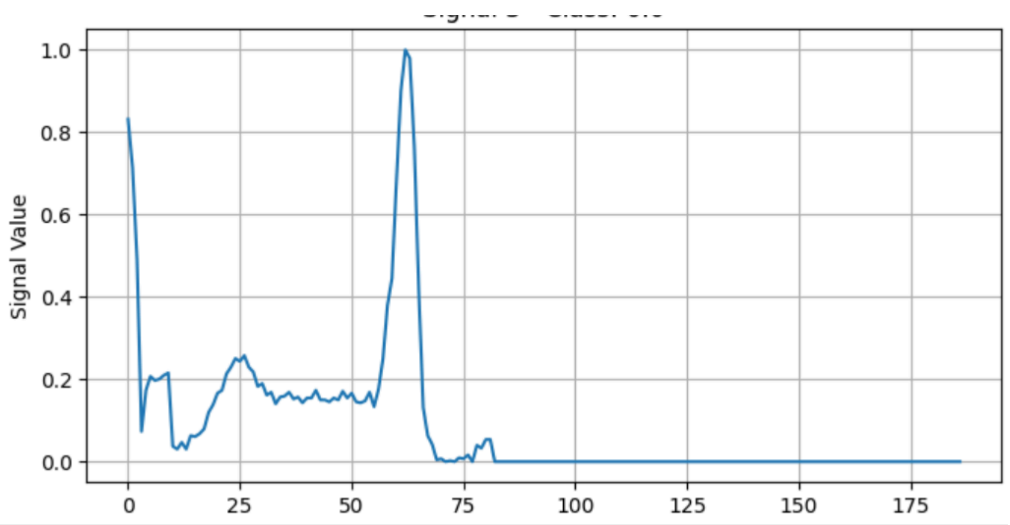
Report kaggle competition of neural networks

Daniel Toribio, Alejo González and Andrés Navarro

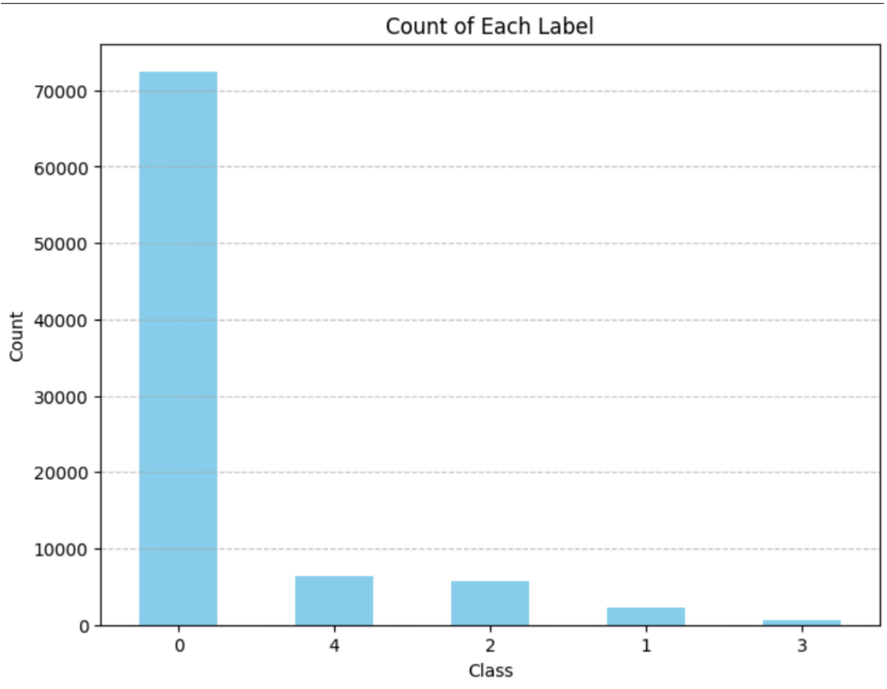
Preprocessing

In the preprocessing part we faced different problems. The first problem was dealing with missing data. For this problem we had different solutions: we can replace the missing values with the mean, median or mode or with interpolation. For the last one we tried the linear and the polynomial one with degree 2. The following images are an example of the result of applying linear interpolation:





Another problem we had is the unbalanced data, that can be seen in the following image:



Here we have two solutions, oversampling and undersampling. For the first one we have tried SMOTE to increase the observations of the rest of classes, but we have seen that we are obtaining new samples that are almost identical to the original ones and we are adding a lot of noise. For the undersampling, we are discarding observations in all classes until we obtain the same number of observations as class 3. With this solution we are losing a lot of information. In conclusion, we have used a combination of both to benefit from the advantages of both sampling.

We have also scaled the data. We had different options but due to the zero padding we had only considered maximum absolute value for scaling. In this way, zero values are not changed and not considered for the scaling.

We tried more things like smoothing the signals or removing outliers but we saw that the performance of the models were not improved.

Models

As a first approach we have implemented a random forest model with sklearn and we obtained good results, (f1 score = 0.86678) but not as good as the ones we have obtained when we use deep learning.

For the deep learning models our idea was to apply one dimension convolutional neural networks, because we have seen in class that they work well in temporal signals. We have also thought that as the data is a temporal series we could apply recurrent neural networks, for example a LSTM. Both networks performed better than random forest, but we thought that it could be a good idea to combine both. This is because by stacking these layers sequentially, the model first extracts local features using the CNN layers and then leverages the RNN layers to model the temporal dynamics and dependencies within the feature representations learned by the CNNs. With this model we obtained one of the best scores (f1 score = 0.87895)

Then we tried the same idea but first the RNN and then the 1D CNN. The result was slightly worse than the previous one (f1 score = 0.87861)

Our final model was to put the 1D CNN and the RNN in parallel, so in this way we avoid having a large neural network where the gradients are not as evanescent as they could be. The score is (f1 score = 0.87493)

We have fine tuned all neural networks parameters, for example, the learning rate, the dimensions of the layers, the dropout probability… , and we have obtained that the learning rate should be around 0.001, the 1D CNN of two layers with an output of 64 channels and another layer with an output of 128, for the LSTM layer the best hidden size was 128 with 3 layers and a dropout probability of 0.4