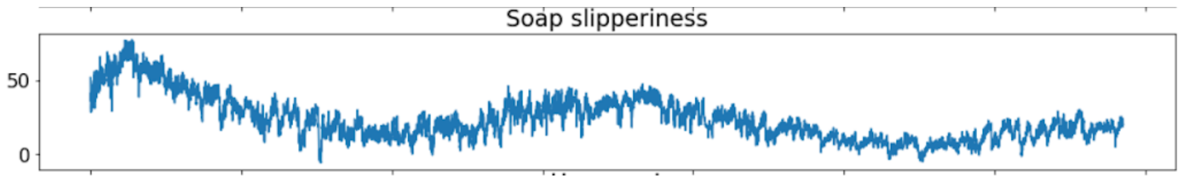
Report Challenge 2 ANN&DL

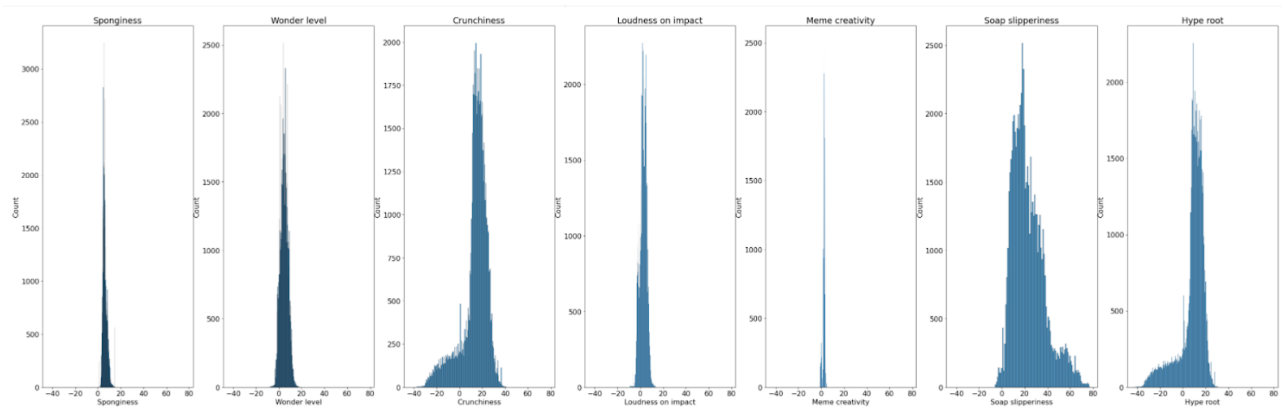
i_Cesaptroni: samuelepartacini, AlessandroMessori, DavideMangano

Exploratory Data Analysis

We started by performing some Exploratory Data Analysis on the dataset. First, we plotted the time series in order to verify if there were any regularity or periodicity in the features trend that we could exploit. Despite at first glance it didn't seem so, by analyzing each feature independently and using different plot resolutions we discovered that most of the features exhibited a periodic pattern of around 100 time steps. We then used this value for our sequence window (we also tried different values but this resulted to be the best one).

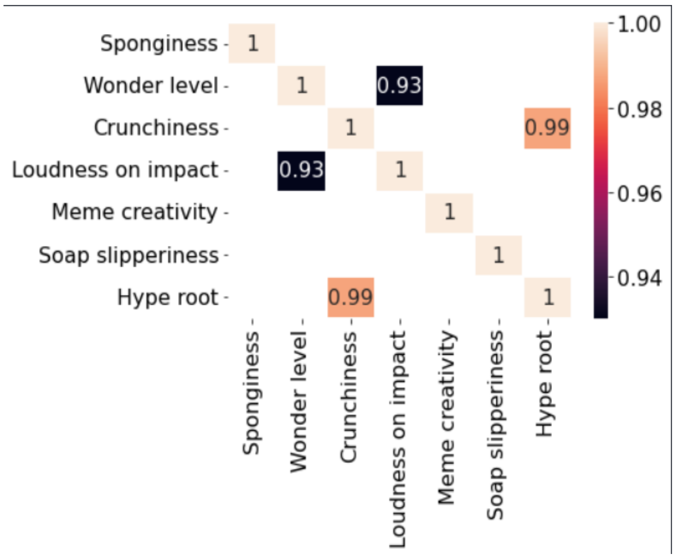


After that, we extracted the statistics of each feature (min, max, mean, std) and we plotted their values distributions. This pointed out how different the range of values and variance of each feature were from each other, and this was one more reason to apply normalization before training.

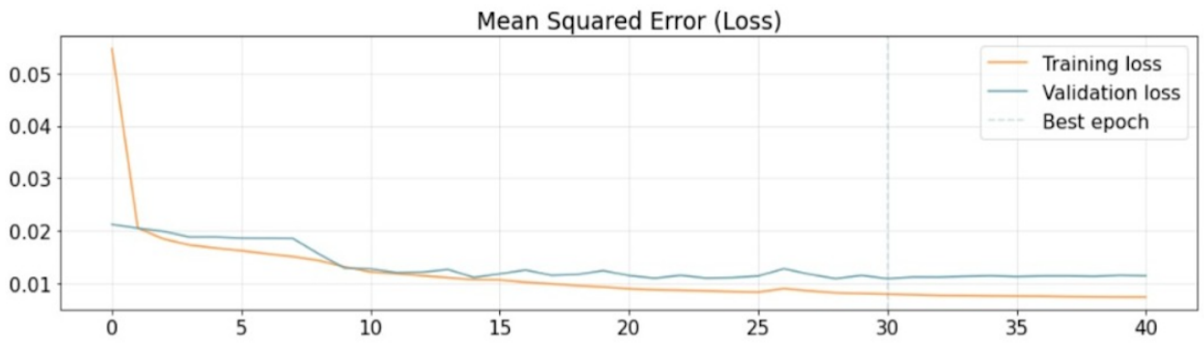


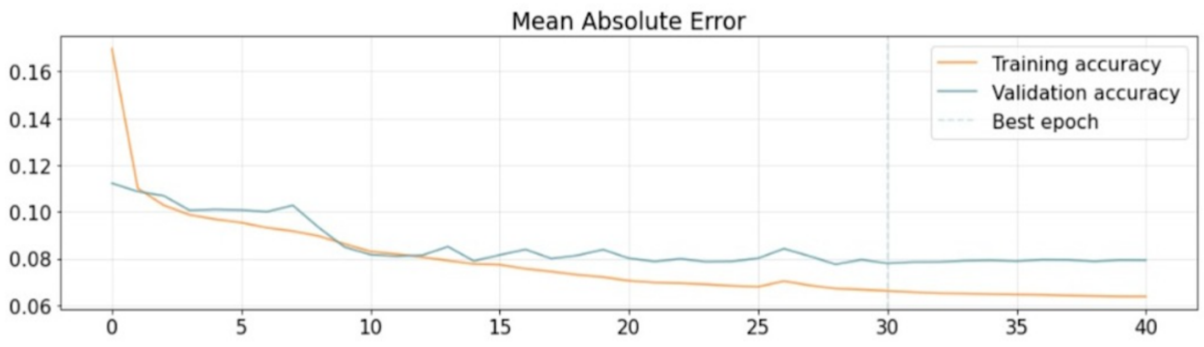
We then plotted the correlation matrix in order to check whether we could find some features with a high correlation that could be deleted in order to decrease the complexity of our model.

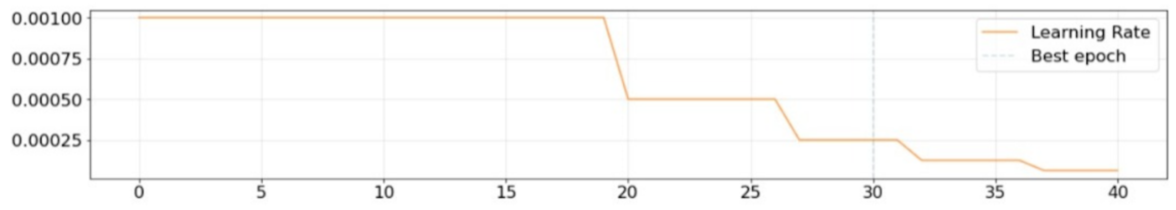
As shown in the picture, we filtered the matrix by plotting only the pairs of features with a correlation value higher than 0.7, and we discovered that Crunchiness had a very high correlation with Hype Root and the same goes for Wonder Level and Loudness on Impact, so these features were candidates to be eliminated in the preprocessing step.
On the other hand, the other pairs of features all had a very low (< 0.7) correlation and so were all important inputs for training the network.



Model Development
We began our trials from a model composed of two bidirectional LSTM layers interleaved by Conv1D layers and MaxPooling. This was followed by a GAP and a Dropout layer to reduce overfitting. At the end we placed a dense layer to generate the output multivariate predictions. We applied to the output a final Conv1D layer with 7 filters 1x1 in order to maintain the output dimension. We used Relu as activation function for the Conv and dense layers. For the training we used a 10% validation split and applied Early Stopping to contrast overfitting, plus the ReduceOnPlateau technique in order to dynamically adjust the learning rate and so improve performance. Initially we trained our models in an Autoregressive manner, predicting one sample at the time and this led us to a RMSE of about 7.01 on the test set. Later we switched to a direct prediction approach that greatly improved our performance. Then we experimented with some different architectures for the network, mainly by adding another dense and dropout layer and testing different values for the dropout.
Since we discovered in the EDA that there was a very strong correlation between Hype Root and Crunchiness, we decided to delete the former from the input variables, in order to reduce the complexity of the model and the overfitting. With this new input shape we managed to slightly improve the performance of the model on the test set. For the same reason we also tried to delete Loudness on Impact, but this time we didn’t manage to improve the performance wrt to the model with 6 input features.
As said before, we started with ReLu which already gave good results. But when we discovered the existence of Swish (more precisely, SiL), which is a smooth, non-monotonic function that consistently matches or outperforms ReLU, we decided to give it a try. Thanks to it, our model performed better on Loudness on Impact and we have been able to improve our RMSE of nearly 0.1 (see the Ensemble Model section for the details). You can find our final model, trained on 6 features, in the notebook ‘modelswish’. These are the related training plots:







Hyperparameter Tuning
During the challenge we did a great effort on finding the best hyperparameters values, starting from the window size to use from predictions. We started by using large values of the window, but then by better inspecting the time series plots we reduced it to 100; just this change brought our test set RMSE from 7.73 to 7.01. Then, as said before we obtained further improvements by switching from auto-regressive to direct prediction mode, obtaining a score of 4.20. We then tried different values for the Dropout layer, obtaining the best results for 0.55.
We also tried to add an additional dense and dropout layer, hoping that by increasing the dropout values the model could benefit from the added complexity without overfitting too much, this new model performed better on some output features but had an inferior overall performance wrt to the one with only one dense layer.
Finally we tried different proportions of the validation split, and we got the best performance with a value of 10% (which makes sense since we are trying to predict time series data and with a low validation split we manage to train on the most updated data) .

Ensemble Model
After trying many different models, we noticed that some of our models performed better on some of the output features and worse on others. We decided then to use an ensemble approach, by training more models different from each other (by the number of input/output features, parameter values and network structure), in order to fine tune each model parameters on a specific output feature rather than trying to train a single one to predict them all.
Initially, we tried to implement the ensemble approach by loading 5 of our models and by predicting each output feature as the mean of the outputs of all models.
This approach didn’t increase our overall performance, so we decided to have each feature predicted directly by the model who had the lowest RMSE on that specific feature based on the results obtained on the test set (this code is found on the model.py file)
By experimenting and adding different models to our ensemble we were able to get a score of **3.6441**.
(We attached only the notebook that had the best performance singularly; including the other components of the ensemble is irrelevant as they only differ from it by the values of some parameters and the input shape).

Alternative trials
During the challenge we tried also to modify structurally the main model. First we tried a ‘clean’ LSTM network (you can find it in the ‘basic_bi_lstm’ notebook), which means we discarded the convolutional and pooling layers while maintaining two bidirectional LSTM layers followed by Dropout and two dense layers (we added one). This model had a local performance similar to the main one (0.012 validation loss), but a very worse performance on the test set (14.9). This was probably due to the overfitting caused by the additional dense layer.

Then we tried a different architecture, similar to the main one, where we placed two (we added one) convolutional layers of dimensions 256 and 128 respectively after the second bidirectional LSTM layer (you can find it in the ‘bi_conv’ notebook). Our intention was to reduce the feature maps dimension in order to reduce the overfitting caused by the last dense layer. As expected, the local performance worsened a little (val. loss of 0.033) but the performance on the test set improved (8.4). Nevertheless, this was still much worse wrt the performance of the main model.

As a last structural-change trial, we replaced the LSTM with GRU (you can find the model in the ‘6-features-GRU’ notebook). Differently from the previous two attempts, at this time we had already discarded the ‘Hype Root’ feature, so the training was conducted on the remaining 6 features. Unfortunately we obtain a slightly worse performance both locally and on the test set (4.26 wrt 4.06 obtained with the same model but using LSTM).