We started as usual with a baseline model. We decided to use 2 stacked LSTMs (32 and 64 units) with bidirectional connection, a global average pooling layer before the final dense layer with ReLU as activation function. We didn't have a clear view of how to choose window and stride's sizes, so we chose 1000 for the window because in our mind in order to predict 864 points the model needed to have as input at least a similar amount of points. We trained this model in two ways: with and without the use of auto-regression technique. With our first submission we got very bad results (around MAE=45). After some code analysis we find out that we were post-processing the data in a wrong manner, so we fixed this and we got way better result (around MAE=8 with both models).
We saw that the auto regression one was slightly better than the other so we decided to follow this technique for the next models. Analyzing the predictions' plots, we saw a flat trend of the prediction, with really small oscillations (while the original signals have a lot of oscillations with high amplitudes). Probably this was due to the high value we gave to the stride. Moreover we detect a strange behavior in the plots: at certain points it fall to very low values like an inverse Dirac delta function (see '*baseline/predictions_plots'*). Probably this behavior was due to the ReLU function that we used as activation for the output layer. Only later on we find out this and we change it.
After this we decided to explore the possibility of using some convolutional layers together with the LSTMs. We started by adding a single 1D-convolutional layer with 128 filters and an average pooling layer before the 2 LSTMs. With this model we got worse performances. Probably this was due to the fact that the features extracted by the convolution-pooling were not good enough. So we decided to stack three convolutional layers (with 32, 64 and 128 filters respectively) before the 2 LSTMs: with this we improved the model performances obtaining a MAE=5.7. We think that the multiple convolution with a small filter size (5) were capable of extracting better temporal features and indeed the plots were no more flatten but we saw oscillation that looked like the training one (probably the small filters sizes and the multiple convolutions compensated the big stride size -20- that didn't allow the model to capture small temporal variation of the signals). Plots of this model can be seen at *'convs_before_lstms/predictions_plots'*.
After we tried to alternate convolutional layers, average pooling layers and LSTMs: indeed we created a model where we repeated this structure three times with an increasing number of filters and LSTM's units (16, 32, 64), ending with the usual global average pooling, a dropout layer and the dense layer. With this model we achieved better results , with MAE=4.5 (see *'3_convs+lstms/predictions_plots'*)

We experimented various models trying to improve the last one, changing some parameters value, but we didn't get improvements. We were not sure if the convolution layers were really useful. Indeed in our mind the LSTMs alone should be enough to capture the time variability of the signals. After some reasoning we convinced ourselves that stride size was too big to allow the model to learn small and rapid variation of the signals.

So we decided to come back to the original model (2 LSTMs) and reduce the window and stride sizes respectively to 200 and 4. Moreover we changed the output activation function from ReLU to sigmoid: in this way we eliminated the strange behavior we mentioned before (it can be seen in *'2lstms_reduced_stride/predictions_plots'*). Furthermore we switched again the model to the one without autoregression since we were not sure that the autoregression really improved the performances (we had only small improvement in the baseline model at the beginning). With this model we got MAE=4.2.

Meanwhile we were trying to improve performances by tuning some parameters values, we were researching how to implement a convolutional 2D layer before the LSTMs. Indeed we wanted to see if it could capture some dependences between different signals, because when we did data inspection we saw a strong correlation between signals 'Wonder level' and 'Loudness on impact', and 'Crunchiness' with 'Hype root'. After some struggle to make it work, we trained the model. Unfortunately just looking at the plots we realized that the results would have been really bad and we decided to not send it to be evaluated (see plots in *'conv2D/predictions_plots'*).

The hyper-parameters grid search brought us to choose this model: 2 LSTMs with 40 and 70 units, window equal to 180 and stride of 3. With this we got a MAE=3.65. Finally in the last week we made some reasoning about all the steps that lead us to the last model and we understood that there could be some other aspect that we could improve. A thing that was not convincing us was the fact that the predictions seems too smooth while the original signals didn't. Our hypothesis was that the cause of that was the average pooling layer. So we switched to the use of a flatten layer after the 2 LSTMs: this increased the model complexity but as always we used early stopping in order to prevent the overfitting. In addition to that we reduced the stride to 1 and more important we change the activation function of the output layer neurons to a linear one, because we were facing a forecasting problem that can be seen as a regression problem so the linear function looked like a better choice. Finally we changed the loss function from MSE to MAE because this is the metric we are using in the competition evaluation (and we have seen

that generally it is a good metric for regression/forecasting problems). With this model we reached the best evaluation with MAE=3.31. Plots of this model can be seen at *'best/predictions_plots'*.

Some final considerations:

- In all the models we have tried to add a dropout layer but we got worse results and maybe it wasn't necessary because we were already using early stopping to prevent overfitting
- With all the models we used min-max normalization as preprocessing. We also tried a different kind of normalization, using the standard normalization (zero mean and unit variance) but we got always significantly worse predictions.
- We didn't mention before but obviously at the beginning we performed a data inspection to analyze and visualize signals, their correlation and their spectrum frequency (see *'data_inspection.ipynb'*)

Moreover we defined multiple functions in the file *'utils.py':*

- *callbacks(…)*: this function returns all the necessary callbacks used for the training (early stopping and reduced LR on plateau) and create parametrized folders for saving checkpoints during training
- *metrics()*: it returns all the metrics (MAE, MAPE, MSE,MSLE, RMSE) that are used also for plots
- *plot_and_save(…)*: this function is used after the training is ended and, after saving the model, it uses the history to plot training and validation metrics and create a parametrized folder to save them as images
- *processing_and_predictions(…)*: it takes the original dataset and preprocess it (with min-max or standard normalization), then it compute the prediction using the model (it can accept both model that uses or not autoregression technique), and finally post-process the data in order to have a variable that can be used for plotting the predictions (using the function *inspect_multivariate_prediction*)