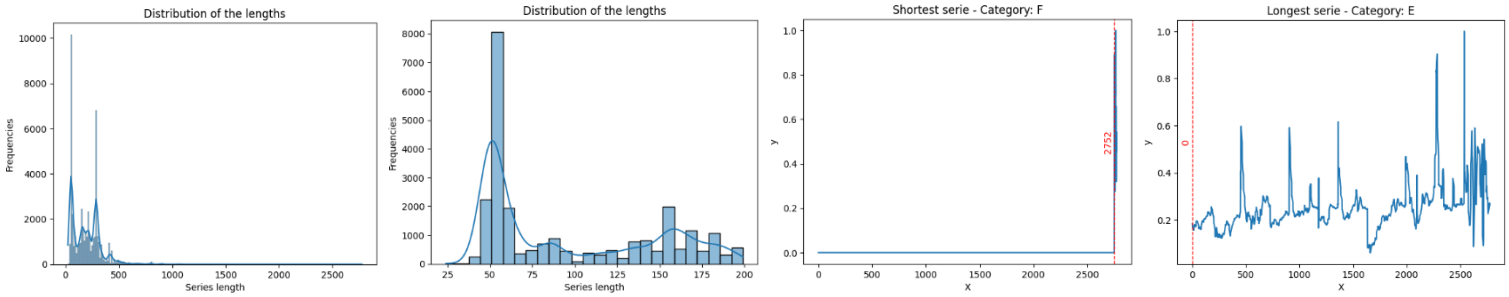
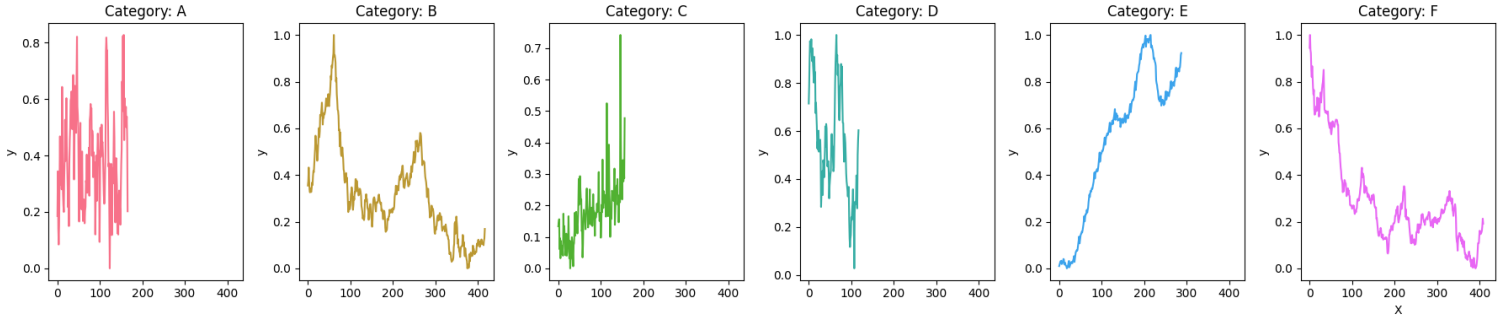
DATASET INSPECTION

The task for this second challenge is time-series forecasting. The provided dataset contains 48000 time series, all 2776 data points long. This length is due to padding, applied to make the dataset easily loadable. However, the actual length of each time series varies according to the intervals provided along with the dataset. Below, we present the distribution of these lengths, and also we provide two limit examples.



Moreover, each time series is associated with a category. There are 6 categories (A, B, C, D, E, F), each representing a different domain.



PRE-PROCESSING

We made various attempts at pre-processing the dataset. Initially, we noticed the presence of some duplicates (26 time series), some even with different assigned categories. Next, we checked for outliers. Initially, we considered to remove some excessively short time series as they would be extensively padded with zeros, potentially introducing bias. We experimented with different length thresholds but ultimately decided not to discard any samples. Finally, we searched for potential constant signals. To do this, we fitted a linear regression for each time series (a line that approximates the series). Subsequently, for each time series, we extracted the slope (m), intercept (q), and MAE. With these three informations for each time series, we sorted them in ascending order based on MAE and slope, the slope was considered in absolute value, to isolate the constants signals. While this method might not be orthodox, it allows the identification of constant signals, i.e., those well approximated by a constant line. This approach allowed us to exclude the presence of constant signals in the dataset. At the end of the pre-processing, we kept 47974 series.

SEQUENCES BUILDING

For the dataset construction (train and validation split), we decided to start by stratifying the dataset split based on category. Both the train and validation sets will thus contain approximately the same distribution of time-series from each domain. This seemed appropriate to avoid biased evaluation or the fact that the model could be trained without “seeing” a specific category. The distributions of categories in both the sets are (approximately): A: 11.94%, B: 22.87%, C: 20.86%, D: 20.87%, E: 22.87%, F: 0.58%.

Then we proceeded with constructing the windows, following the specifications provided by the problem, which involved creating windows of size 200. Additionally, as a design decision, we immediately considered a sequence length to predict of 18 (which will be simply cropped during the first phase of the challenge). In doing so, we aimed to minimize sample waste. To achieve this, we calculated the minimum padding in such a way that the time series could be evenly divided, considering the window size and the stride (we choose 10), as padding method we decided to replicate the signal when it was shorter than a window, while when we have to slight adjust the length of the signal to match the stride we padded with zeros. After completing this phase, the dataset is composed of 212982 windows in train and 54912 in validation.

MODEL INSPECTION

For the model selection phase our approach was to try as many models as possible, in particular, the approaches we tried are:

- Bidirectional LSTM + Attention
- Bidirectional LSTM + Attention + Conv1D
- ResNet 1D
- ResNet 1D with Convolutional Block Attention Model (CBAM)
- Feed Forward Neural Network
- Transformer
- Time2Vec
- Various ensemble strategies

Right from the start, while training the models (window = 200, telescope = 18), we realized that the task was far from simple. Although we achieved decent performance, almost no model (except for ResNet) could overfit the dataset. From this list of models, we immediately discarded the Transformer as it didn't outperform the other models and also it was computationally expensive. The failure of the Transformer didn't surprise us, primarily because we didn't have an appropriate embedding for the time-serie and also because the Transformer is time-invariant, thus its self-attention mechanism isn't particularly effective in this case. The architectures we decided to pursue are:

- Bidirectional LSTM + Attention
- Bidirectional LSTM + Attention + Conv1D
- ResNet 1D with Convolutional Block Attention Model
- Feed Forward Neural Network

BiLSTM + Attention

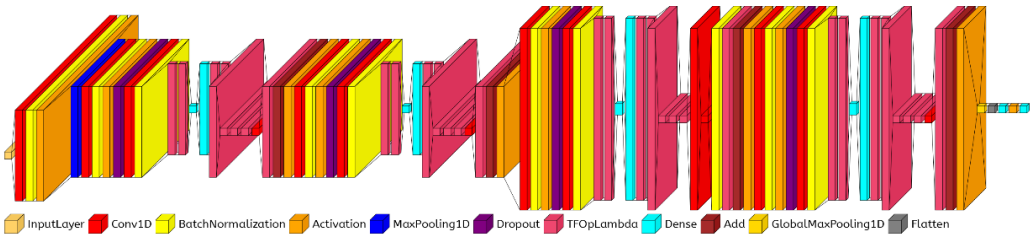
One of the simplest models we tested, it consists of a BiLSTM with 256 units followed by a Luong attention layer, followed by 2 dense layers with 64 and 32 neurons (ReLU activated), and finally an output layer with 18 neurons with linear activation. Despite its simplicity, this model has shown good performances on our validation set (MSE = 0.0068, MAE = 0.052) and has not exhibited any particular signs of overfitting.

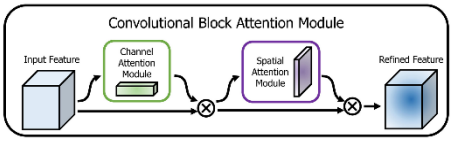
BiLSTM + Attention + Conv1D

Another architecture we tested is the combination of a Bidirectional LSTM and a CNN. We attempted this architecture because in literature, we found many publications that discuss similar architectures. The network consists of two LSTM layers, each with 128 units, followed once again by a layer implementing Luong attention, followed by a sequence of Conv1D and MaxPooling1D layers. With this network we archived MSE = 0.0070 MAE = 0.053 on our validation set.

ResNet1D CBAM

The most complex architecture we tried is the ResNet 1D with the addition of the Convolutional Block Attention Module (CBAM) (Sanghyun Woo, 2018). The Convolutional Block Attention Module (CBAM) is an attention module designed for convolutional neural networks. Given an intermediate feature map, the module sequentially deduces attention maps along two separate dimensions, channel, and spatial. These attention maps are then multiplied with the input feature map to adaptively refine features. With this network we archived MSE = 0.0077 MAE = 0.056 on our validation set. The architecture is quite intricate and is detailed below:





FFNN

For the last architecture, we tried a simple FFNN. While this might seem like an unconventional approach, in reality, it's often compared in literature with recurrent networks and LSTMs. Regarding performances, are not significantly different from the other architectures, but it's computational efficiency is superior. The network consists of 8 hidden layers following a decreasing number of units (pyramidal network), ranging from 2048 to 32, all using the ReLU

activation function. The output layer comprises 18 neurons with linear activation. With this network we archived MSE = 0.0079 and MAE = 0.057 on our validation set.

HYPERPARAMETER TUNING

During this phase, we looked for the best parameters for the four models (**BiLSTM+Conv1D, ResNet1D+CBAM, FFNN, Standard LSTM**) by using the grid search algorithm. This allowed us to test all combinations within a parameter grid. As can be seen from the results, almost all the models exhibit very similar performances.

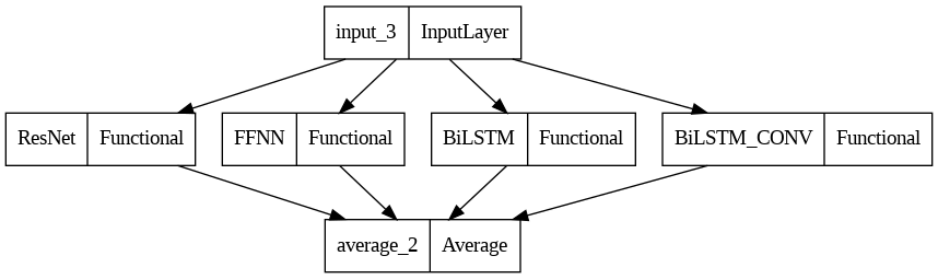
	BiLSTM+Conv1D	ResNet1D+CBAM	FFNN	Standard LSTM
MSE	0.0070	0.0075	0.0074	0.0068
MAE	0.0533	0.055	0.055	0.0520

The hyperparameters tuned were the batch size and the minimum learning rate to use in the learning rate scheduler, In the table below we reported the tuned hyperparameters for each model:

	BiLSTM+Conv1D	ResNet1D+CBAM	FFNN	Standard LSTM
Batch size	64	64	256	64
Minimum L.r.	1e-8	1e-8	1e-5	1e-8

ENSEMBLE

Since we weren't entirely satisfied with the performance of individual models, we opted to create an ensemble model using the tuned models previously trained. We simply employed an Average layer, which calculates the average of the models' predictions. With this strategy, we observed a substantial improvement in performance.



With this model we archived MSE = 0.0061 and MAE = 0.049, on our validation set.

TESTED VARIATIONS

In this section we briefly summarize further trials. The first attempt to increase the performances consisted in training models with an autoregressive approach, although this method did not yield better results than a single prediction approach. Additionally, other model variants were utilized, including a BiLSTM-Convolutional architecture proposed (Yongbo Liang, 2020), VGG 1D, a bagging approach comprising 15 LSTM regressors, Time2Vec (Seyed Mehran Kazemi, 2019), an ensemble with six models one for each time-serie category and also a custom ensemble layer was attempted, weighing the predictions of the networks. None of these attempts resulted in performance improvements. We also tried variations on the dataset as for example zero padding, different strides and also data augmentation with SMOTE (N. V. Chawla, 2011).

RESULTS AND CONSIDERATIONS

At the end of the challenge, our final model is an ensemble composed of 4 models: a BiLSTM, a BiLSTM+Conv1D, Resnet1D+CBAM, and a FFNN. The combination of these models occurs through an averaging layer. The performances achieved by this model at the end are satisfying, particularly because they are consistent across the validation set, first test set, and second test set. Consequently, we can observe that the model demonstrates good generalization capabilities across the data. Below, we present a comprehensive table of the performances of the final model.

	Validation (18 Samples)	Test 1 (9 Samples)	Test 2 (18 Samples)
MSE	0.0061	0.0053	0.011
MAE	0.049	0.051	0.072

The performances on the second test set are computed by retraining the model on the whole provided dataset so both on training set and validation set.

References

Chollet, F. (n.d.). keras. Retrieved from https://keras.io/

N. V. Chawla, K. W. (2011). SMOTE: Synthetic Minority Over-sampling Technique. Retrieved from https://arxiv.org/abs/1106.1813

Sanghyun Woo, J. P.-Y. (2018). CBAM: Convolutional Block Attention Module. Retrieved from https://arxiv.org/abs/1807.06521

Seyed Mehran Kazemi, R. G. (2019). Time2Vec: Learning a Vector Representation of Time. Retrieved from https://arxiv.org/abs/1907.05321

Yongbo Liang, S. Q. (2020). Deep Learning Algorithm Classifies Heartbeat Events Based on Electrocardiogram Signals. Retrieved from https://www.frontiersin.org/articles/10.3389/fphys.2020.569050/full

CONTRIBUTIONS

Andrea Bertogalli	***Niccolò Balestrieri***	***Nicolò Tombini***
Pre-processing: 60%	Pre-processing: 20%	Pre-processing: 20%
Model tuning: 20%	Model tuning: 60%	Model tuning: 20%
Report: 20%	Report: 20%	Report: 60%
Notebook refactoring: 33.3%	Notebook refactoring: 33.3%	Notebook refactoring: 33.3%

**NOTE: all team members worked on all parts of the project.*