# Time Series Forecasting Challenge
## Artificial Neural Networks and Deep Learning
**Team: GeoPalle**

**Lorenzo Carlassara, Ellen Poli, Matteo Gobbi Frattini, Zhongyou Liang**
*Politecnico di Milano, Geoinformatics Engineering*

## 1. Data Exploration

The proposed challenge is about time series forecasting. The goal of the task is to understand the trend of the time series in order to forecast on the first future samples. The hidden test set is composed by 60 time series of length 200 in a first phase and by all the samples in the final phase. [3] [4]

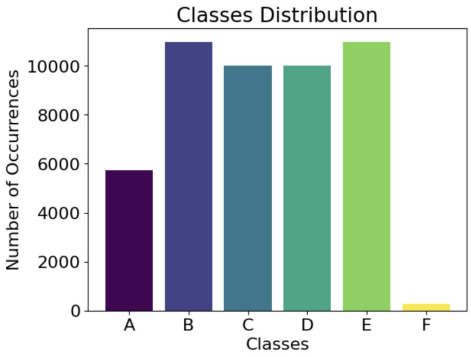


Figure 1: *Classes distribution: please note the disproportionate unbalancing between class F versus the others*

The provided dataset contains 48000 time series, each of which has 2776 values. In order to ensure that they are of the same length of 2776 a padding have been applied since the architectures required as input series with the same length. The time series provided covers six different domains. With a preliminary analysis, we found out that the class distribution is unbalanced, as shown in Figure 1.

## 2. Preprocessing

In the initial phase, we face challenges with the dataset's size, making it impractical to process all at once. As a result, we explore various techniques to ensure both feasible training time and memory usage.

The first attempt involves converting the data type to float32, enabling us to utilize the entire dataset. However, this comes at the cost of reduced precision compared to float64.

Another discarded approach, in an effort to maintain precision without sacrificing computational efficiency, is to exclude time series shorter than the predefined window length (200), along with the telescope (9 in the initial phase) the stride (10, 20, 50 or 100) and test size (209). Also we pay attention to retaining time series from all classes, ensuring that information loss is minimized and that the class proportions remain consistent with the original distribution.

Even though this refinement allows for a more manageable dataset while preserving essential information, the amount of data remaining for the training is reduced a lot.

### 2.1. Class Imbalanced

To face the problem of the unbalanced class distribution, we decide to try an undersampling approach. With only 277 time series, class F has the lowest class distribution, as can be seen by looking at the distribution of classes shown in Figure 1. To avoid that, the model learns mainly the trend of the bigger classes, we apply an undersampling on the classes. We keep just 277 time series for each class as the number of time series of the smallest class (F)(Figure 4). As we can see from Figure **??** also the average length

of the time series for each class is homogeneous. This procedure allows to significantly decrease the loss value. [5]

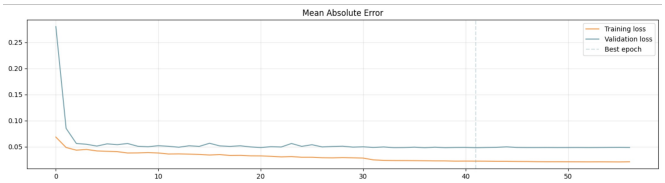


Figure 2: *Encoder-decoder model, 1st phase leaderboard: [MSE=0.00692684, MAE = 0.05990572]*

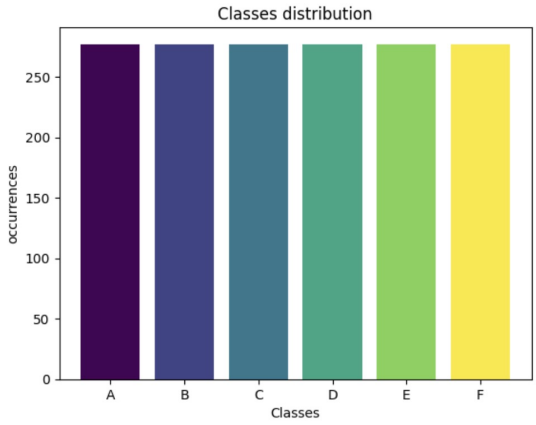


Figure 3: *Constant classes distribution after keeping 277 time series for each class.*

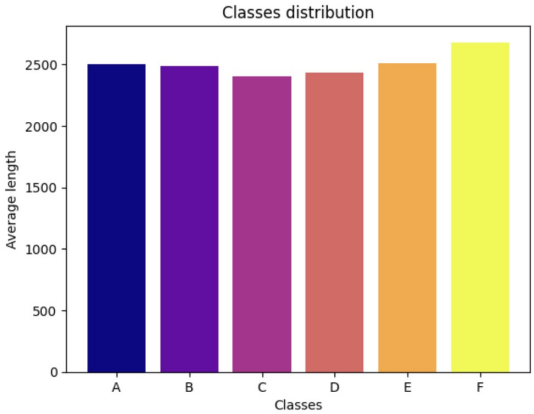


Figure 4: *Avarage length of the time series for each class: A: 2504, B: 2487, C: 2405, D: 2435, E: 2507, F: 2678*

## 2.2. Sequences

We start trying the following values for building from the timeseries the sequences to feed the model:

- window = 200
- stride = 50
- telescope = 9

In particular, we use a not-too-small stride to avoid occupying too much memory. As the dataset's dimension drastically decreases following the class balance, we train the model with a smaller stride (tested values: 10, 20) in order to increase the accuracy of the input trends description.

## 2.3. Dataset Splitting

We prefer to employ all the data to train the model instead of splitting the dataset into a training set and a testing set. Therefore, the testing phase is performed directly in codalab with the hidden test dataset. One thing we notice is that during Phase One, there is a little positive gap between the best validation loss obtained in colab and the one provided by codalab test. In addition we choose to address class imbalance through undersampling the classes. Undersampling is commonly used when there is a significant class imbalance, as in our case, to ensure that the model receives balanced representation from each class during training. [5] This involves maintaining the same number of time series for each class, focusing on the longest time series without cutting out data in time.

Furthermore, we discard the implementation of a RobustScaler since it does not provide any performance improving.

## 2.4. Masking layer

As final step of the data preparing, we decide to introduce a masking layer since the dataset has a high number of zeros. For each timestep in the input tensor, if all values in the input tensor at that timestep are equal to 0, then the timestep is masked (skipped) in all downstream layers. The masking layer helps the model recognise and ignore the padded zeros during training and focus only on the relevant parts of each sequence. The masking layer can be leveraged to optimize computation. For example, it allows the

model to skip unnecessary operations on zero values, making the processing more efficient.[2]

## 3. Architectures

Here we provide, in chronological order, a brief insight of the main 1-dimension models structure:

1) **pure RNN**, as seen during the practice lectures, is made of only the LSTM layer proposed by Hochreiter [1], and the essential layers required to process the train batches for the specific homework. The purpose of this simple network is to provide a gentle first approach to the timeseries.
2) **LSTM + CNN** is introduced as soon as we realize that the pure LSTM has not enough parameters to model the timeseries withouth underfitting.
3) **seq2seq** comes naturally with the need of encrease the model.
4) **Bidirectional LSTM with Encoder only** is a try we consider just to test the single contribute provided by the two components to see if a stand alone was a possible idea.
5) **Bidirectional LSTM + ResNet-style CNN** is taken into account in a spontaneous way, and then implemented to repel the risk of the vanishing gradient after the decision to encrease even more the model complexity

During the 1st phase of the challenge the lower loss is obtained with an homemade Seq2Seq networks. Whereas, during the 2nd phase, trying to combine the properties of the ResNet with the structures of an inspired Encoder-Decoder we notice very similar performances despite of the different architectures.

### 3.1. Performance

We choose to monitor the Mean Absolute Error (MAE) as the loss metric to provide a more resilient interpretation of the outcomes. This decision takes into account the individual normalization of the series, presenting the average percentage error (scaled between 0 and 1) observed on the predicted points in comparison to the ground truth.

As mentioned before, we test several models, starting with the simpler ones and working our way up to more sophisticated ones, in order to face the time series forecasting challenge. Our initial model is a standard LSTM model, which doesn't perform so well on the leaderboard. The first successful attempt (at least for us) is a seq2seq encoder-decoder with a bidirectional LSTM right before the bottleneck between the encoder and the decoder after implementing an autoencoder model first. As a result, the input's context from the past and future can be captured by the model. Next attempt, when we train the encoder part without the decoder, we try both to combine it with a bidirectional LSTM and without. At the endof the 1st phase the best results are given by the encoder coupled with the bidirectional LSTM, but it is still a bit worst then the autoencoder with stride 20. After that we try also a 1D-ResNet style architecture. We notice that it doesn't increase the performances with respect to the autoencoder, indeed it provides similar MSE results. We retrain it with different combination of parameters. Moreover, we notice that in the leaderboard with a different split between training and testing The results of the main trial are shown in table

| Model | Stride | MSE |
|---|---|---|
| Autoencoder | 50 | 0.01 |
| Autoencoder | 20 | 0.006 |
| Autoencoder | 10 | 0.008 |
| Encoder + BidirectionalLSTM | 20 | 0.007 |

TABLE 1: *First phase: forcasting on 9 time steps.*

| | |
|---|---|
| Autoencoder | 0.0123 |
| 1D-Resnet style | 0.0131 |

TABLE 2: *Second phase: forcasting on 18 time steps.*

As we expected the MSE values result to be worst in the phase since the forcasting window is longer.

## 4. Conclusion

After conducting numerous trials, we observe that altering the number of convolutional layers or their dimensions do not lead to significant improvements beyond the already attained threshold level of performance. Consequently, we come to the realization that the predominant factor influencing outcomes is the chosen architecture, rather than variations in these specific parameters.

**Contribution**:

1) **lorenzo.c** Lorenzo Carlassara (10601118): design and implement the final neural network of both phases, as well as the pipeline of previous models e.g. Autoencoder and 1dResNet-style CNN. Minor contribution to the report.
2) **EllenPoli** Ellen Poli (10728490): dataset preprocessing, implement and adapt different networks e.g. Autoencoder and Seq2seq. Report writing.
3) **MatteoG25** Matteo Gobbi Frattini (10581031): attempt to implement autocorrelation, not deployed.
4) **zyl** Zonghyou Liang (10905937): adapt pure LSTM with different hyperparameters testing.

## References


[1] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[2] *Masking Layer*. https://keras.io/api/layers/core_layers/masking/. Accessed: 2023-12-7.

[3] *Time series forecasting*. https://www.tensorflow.org/tutorials/structured_data/time_series. Accessed: 2023-12-7.

[4] *Traffic forecasting using graph neural networks and LSTM*. https://keras.io/examples/timeseries/timeseries_traffic_forecasting/. Accessed: 2023-12-7.

[5] *Undersampling*. https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/. Accessed: 2023-12-7.