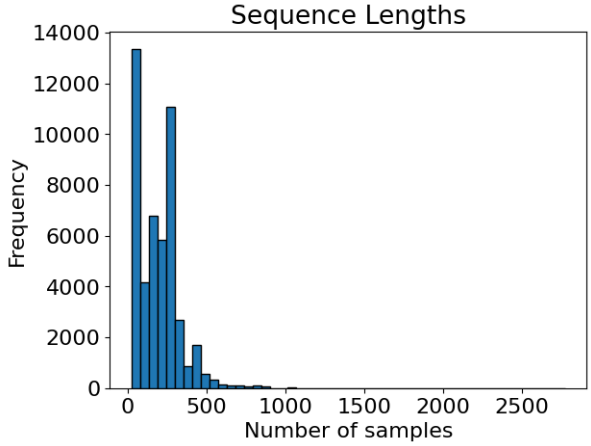
AN2DL Challenge 2

1. Introduction to the Challenge

1.1. Problem highlights

The problem consisted of a sequence-to-sequence problem as given a time series of variable length it was required to predict the next nine future values.

Since each time series had a label, and thus the sequences were organized in categories, it was asked to create a forecasting model able to generalize and effectively perform forecasting independently from the specific category.



1.2. Dataset Description

The dataset provided was composed of 48000 padded sequences of variable length up to a maximum of 2776 values. As can be seen in the figure above most of the sequences had less than 500 samples. The categories in which the sequences were grouped were selected from six different fields, and were not balanced, as is shown in table below. This was one of the main reasons that lead to disregarding class information to create a unified model able to predict future values regardless of the class.

Class	Number of sequences
A	5728
B	10987
C	10017
D	10016
E	10975
F	277

2. Preprocessing Routines

2.1. Visualization, Data Cleaning and Dataset Splitting

The first thing done was the exploration of the dataset. Starting with the plot of random signals, they were subsequently plotted by class and slight difference was noticeable by inspection, among the same class commonalities were found but also similarities were present across different classes. Therefore, a classifier was implemented to try to discriminate sequences according to their class. To further improve it, since the classes were very unbalanced, their weight was adjusted in the model. The reasoning behind this was that achieving a reliable classification, if possible, would allow to spot outliers, which are considered external to all deduced classes, and to prevent from inserting them in the training dataset.

2.2. Hyperparameter Tuning

When investigating a new architecture, the chosen values for the patience of early stopping and learning rate scheduler were respectively twelve and ten. On top of that, for the best performing models, the patience parameter for the early stopping was increased to fifteen, granting a more in-depth investigation. The batch size was 64, the starting epochs

150, and the validation split was set to 0.2. The dropout rate in general was set to 0.25 and for attention layers 0.2 and 0.3 depending on the architecture.

2.3. Sequences Split

The unpadded dataset was split into training and test set, respectively taking 75% and 25% of the samples, according to each time series length. Each element of the two sets was then split into windows of 200 timesteps to be fed to the network. Each window was extracted from the original dataset with a stride of 50, to provide better temporal context. The stride parameter was set equal to 50 after several attempts. It was the best compromise between the length of the generated dataset and the amount of information retained for temporal context, furthermore, allowing for a reasonable training time, due to a lack of excessive redundant computation. Both training and test sequences were padded to ensure that they matched the input size of the network, and in the network a Masking layer was added to avoid the learning of the padding. The shortest sequences (of less than 50 elements) were then discarded, since they were too small to be significant in training and this proved to be an effective strategy to improve the overall performance across all the models. Some data augmentation techniques were experimented with by adding noise to the sequences, but the performance decreased, so it was avoided in the end. Robust scaling of the sequences was also tried but did not yield the desired improvements.

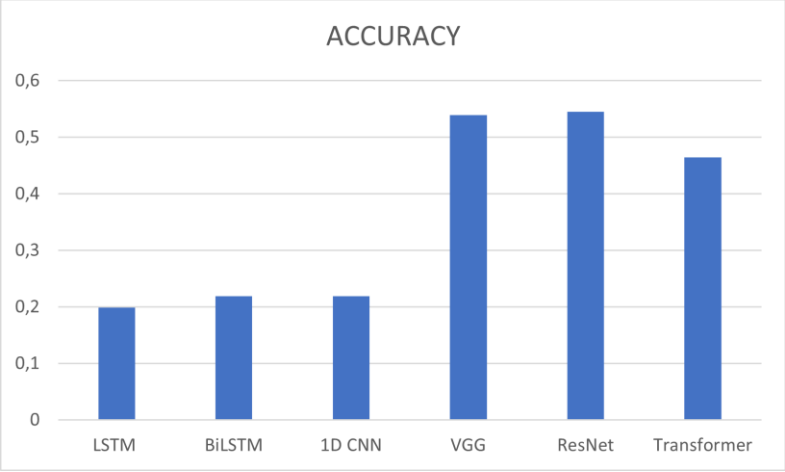
3. Investigated Architectures

3.1. Built-from-scratch Models

3.1.1. Classification

Regarding the classifier the explored models were: Simple Feedforward Network, LSTM, BiLSTM, 1DCNNs with VGG style and ResNet style architectures, and Transformer. The best result was obtained by the ResNet classifier with an accuracy of 0.54. It was decided that the model was not good enough to be implemented since it did not discriminate well between the classes. A further check was performed by feeding the class labels to the forecasting model as well, and some slight improvements were accomplished.

At this point, some work was done in trying to build an embedding of time series using an autoencoder, but it did not bring satisfying results. An enhancement was obtained with the time2vec approach, integrated in some later models.



3.1.2. Forecasting

The forecasting architectures have nine output neurons and make direct forecasting. Autoregressive forecasting was implemented during the second phase of the competition, by first predicting the next nine values for each sequence, then adding them at the end of the test set and using them to make the next prediction. Some architectures implement cropping layers at the end to adjust the output to the desired size. After having prepared the data as described before, the first model tried consisted in a simple Bidirectional LSTM layer, which yielded a test mean squared error of 0.023.

Subsequently it was tried to stack multiple LSTM layers, but this approach revealed itself to be too computationally expensive, since the network was unrolled too many times. Simpler architectures were explored with only one LSTM layer and one 1DCNN layer and an Attention layer, and they showed some slight improvement.
The Attention layers explored were: Simple, Additive and MultiHead. The overall best performing layer was the MultiHead Attention layer, which was integrated in all the models from now on. As the exploration of deeper models began, the architecture considered was assembled with two parallel cascades, one of LSTMs and one comprehending convolution layers. Attention layers were inserted in the model. This approach was not improving the previous result and we decided to combine the results of the two cascades by inserting some residual connections between the recurrent part and the convolutional one. The model performed very well, with a test mean squared error of 0.019 and a 0.0065 on the platform. Another deeper approach consisted of stacking multiple convolution layers, then adding an LSTM, then again, more convolution layers and finally a dense network to perform the forecasting. This model performed moderately well, however the previous one was to be preferred.

3.2. Transformer

The last architecture experimented with was the transformer. After having implemented it and let it train, it yielded our best results, scoring a test mean squared error of 0.017 and a 0.0056 on the platform. Wanting to improve even further we coded an enhanced version of the previous transformer that also used classes. The input of the classes was concatenated after the global average pooling to be directly fed to the last dense layers. This model yielded a slight improvement on the previous one and could extract significant information from the classes. It was concluded that the categories were explicative in some way, although not explaining the complexity of the whole dataset.

3.3. Ensemble Architecture

In parallel to the line of reasoning that brought to the transformer it was imperative to verify whether splitting the dataset by length of the sequences was a promising idea. So, the sequences were divided into two classes, the ones shorter than 200 and the ones longer. Then two models were trained on the new datasets, were validated, and were weighed in the ensemble with a normalized inverse of the mean squared error. The ensemble model performed poorly, so it was concluded that this was not an architecture worthy of being developed.

4. Proposed Model

The final proposed model is the transformer with the classes enhancement that performs autoregressive forecasting for every sequence of nine timesteps.