Time Series Classification Report

Artificial Neural Networks and Deep Learning - a.y. 2022/2023

Beatrice Insalata
Codalab Name: B34
Team Name: Personal Trainers
10708628 - 222966

Lorenzo Mauri
Codalab Name: Lorma200
Team Name: Personal Trainers
10696228- 225949

Matteo Pancini
Codalab Name: matt4pa4n
Team Name: Personal Trainers
10656944 - 223085

***Abstract*—The purpose of this challenge was to correctly classify samples in the multivariate time series format, mapping the information contained in their features calculated over time to 12 labels. In this report, we discuss, explore and compare our development and design process.**

I. Preliminary Dataset Analysis

A. Data Distribution Among Classes

Initial dataset exploration was performed by plotting the time series samples and examining them one by one. Given

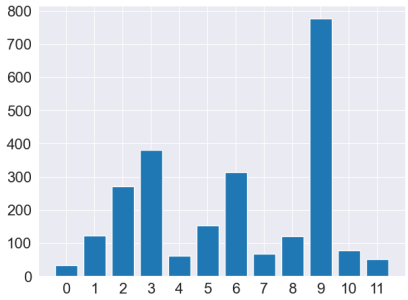

Fig. 1. Dataset distribution of samples for each class shows imbalance in the number of elements per label.

the high structural complexity of data and the scarce amount of information about its features and their interpretation, our research started with preprocessing operations, in order to improve classifier performance.

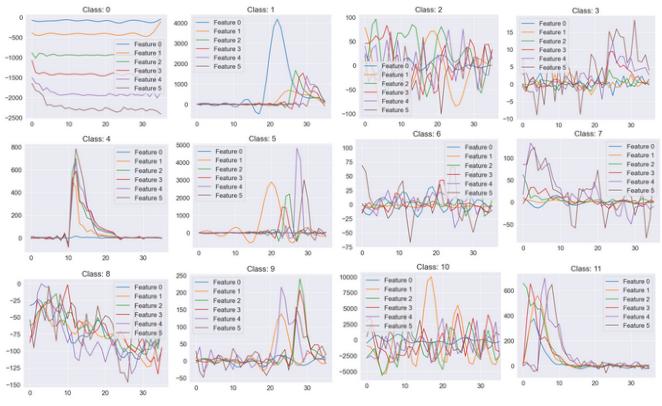

Fig. 2. Plottings of feature trends for each class show distinct behavior and irregularities. .

II. Preprocessing

A. Normalization

Several manipulations were performed to extract more information from the dataset. Initially, normalization was applied to overcome the scale differences between feature ranges.

- **Min-Max Scaling**, applying the function to the dataset to reduce the features' value interval between 0 and 1.
- **Z-Score Normalization,** bringing the dataset's mean to 0 and its variance to 1.

The operations were tested on our 1D Convolutional Network to evaluate their results.

Transformation	Val_Accuracy
Initial Set	0.6584
MinMax	0.6008
Z-Score	0.7192

Although promising in the beginning, none of these operations resulted in a significant improvement over classification tasks. This might be due to earlier dataset processing and changes, which mutated the actual behavior of the time series.

B. Window Resizing

The level of noise in the data and the lack of seasonality suggested that a smaller window size might have been more appropriate. We tried different versions but making the window size too small would produce much worse results. Given that, we experimented with a size of 20 and a stride of 1: this technique led the models to overfit the training set in a few epochs. This could be a possible consequence of the small dimension of the time series.

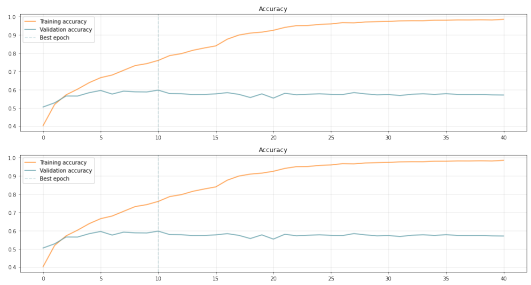

Fig. 3. Final model categorical crossentropy and accuracy trends after window resizing .

The merging of more samples in a single sequence was also proposed but ultimately discarded, to avoid introducing biases or unrealistic variations in the data.

C. Data Augmentation

The limited amount of data led us to consider augmentation as a valid option, generating new samples to deal with present noise and improve generalization capabilities. The operation was performed using TSAUG augmentation function, which applies transformations to the standard time series to add generated entries. We also experimented with different augmentation techniques such as Weighted Dynamic Time Warping Barycenter Averaging (WDBA), discriminative guided warp (DGB), and random guided warp(RGW). We tried this augmentation alone and also combining it with some simpler methods like adding jittering or applying to scale. After some testing we obtained mixed results: with some models, the augmentation produced a small increase in accuracy while, in others, it was counterproductive. In particular, we obtained a 0.02 increase in performance with discriminative guided warp when using a CNN + LSTM model.

Layer Augmentation	Model	Val_Accuracy
TSAUG	CNN + LSTM	0.68
WD	CNN + LSTM	0.68
DGW	CNN + LSTM	0.70
RGW	CNN + LSTM	0.69
DGW	ResNet	0.58

D. Feature Engineering

To diminish the impact of potentially unnecessary or redundant information, we applied some operations on the dataset's features.

- **Feature Removal**, reducing the dimensionality of the set by eliminating one feature or more at a time. The technique didn't produce satisfying outputs.

Deleated Feature	Val_Accuracy
0	0.6358
1	0.6481
2	0.6337
3	0.6235
4	0.6296
5	0.6276
No Delete	0.6564

- **Feature Manipulation** Feature Manipulation, including operations of scaling features by a constant factor and adding weights to correct imbalance in weaker classes.

None of these approaches gave us particularly relevant results.

III. Implemented Models

We explored multiple models and compared their performance in order to determine the best one for the given dataset. The indicators that will be reported have been obtained without any form of data preprocessing, since the final outcomes were found to be negatively impacted by it. Every model was trained after an extensive hyperparameter search with Keras Tuner was performed. The model with the highest score was selected as the best one for each structure.

A. BiLSTM + 1D Convolutional Network

To take advantage of both the models' strengths, we combined a BiLSTM and a CNN: the CNN architecture is able to extract features from the time series data, while the BiLSTM can capture temporal dependencies, producing a potentially improved outcome. Two approaches were mostly adopted:

- **CNN + BiLSTM**: since using a CNN to extract features from the time series data and reduce its dimensionality can result in more efficient processing by the BiLSTM. Our structure consisted of a 1DCNN layer followed by a MaxPooling layer, and 2 BiLSTM layers. To improve generalization capabilities, we introduced 3 Dropout layers, which were positioned in between BiLSTM ones. Overall, the results on the training set, especially after tuning, looked promising, but the model was found to be prone to overfitting.
  In the table below the outcomes of the best models for each architecture after tuning on validation and test set

Model	Val_Accuracy	Test_Accuracy
CNN + BiLSTM	0.7487	0.6896
Concatenation	0.7538	0.6880

- **BiLSTM + CNN Concatenation**: a similar approach to the previous one but with the concatenation of a BiLSTM followed by a more complex layered CNN. The idea was taken from the paper "LSTM Fully Convolutional Networks for Time Series Classification" [2017], by Karim. Majumdar. We explored the base network and executed some structural changes in order to avoid overfitting through Dropout layers. Hyperparameter tuning was performed on dropouts and filters. Even after data normalization, its output was not satisfying enough to make the model our primary choice, underperforming on the online dataset.

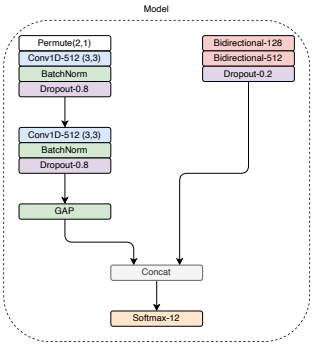

Fig. 4. Structure of BiLSTM and CNN Concatenation model.

B. AutoEncoder

Another explored structure was the autoencoder, given its ability to learn useful features from the data automatically.

It's also capable of handling missing or corrupted data, as they are trained to reconstruct the input rather than predict a target value. We explored different encoder-decoder structures and the best one came out to be the simplest, as the picture below shows. Then, we detached the decoder and connected a classifier composed of a Dense layer and a Dropout layer. This structure has been very promising during the whole challenge, with a peak of 0.7333 Val_Accuracy, but it wasn't able to overtake 0.6955 on the online evaluator.

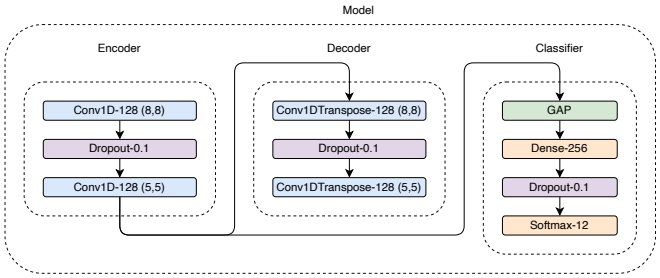

Fig. 5. Structure of Autoencoder type network.

IV. Final Submission Model

A. Model

CNN was the preferred final architecture based on the overall result comparison between models. This kind of network was deemed effective for our data because of the presence of complex patterns and dependencies, but the lack of more samples made it difficult to obtain truly optimal results. This submission was forwarded with no preprocessing on input data, given our exploraiton outcomes. The parameter we used for training were:

- Batch size = 64
- Test split: 0.15
- Validation split: 0.15
- Loss = Categorical Crossentropy
- Optimizer = Adam
- Learning rate: [1e-3;1e-5] + ReduceLROnPlateau

Layer (type)	Output Shape	Param #
Input	[(None, 36, 6)]	0
Conv1D	(None, 36, 128)	6272
BatchNormalization	(None, 36, 128)	512
Dense	(None, 36, 128)	16512
Conv1D	(None, 36, 256)	164096
BatchNormalization	(None, 36, 256)	1024
Dense	(None, 36, 256)	65792
Conv1D	(None, 36, 128)	163968
BatchNormalization	(None, 36, 128)	512
Dense	(None, 36, 128)	16512
Conv1D	(None, 36, 256)	262400
BatchNormalization	(None, 36, 256)	1024
Dense	(None, 36, 256)	65792
GlobalAveragePooling1D	(None, 256)	0
Dropout	(None, 256)	0
Dense	(None, 12)	3084

Total params: 767,500
Trainable params: 765,964
Non-trainable params: 1,536

Taking as initial reference the net shown during the labs, we used a configuraton with 4 Conv1D layers, each followed by a BatchNormalization to make the model more robust to data fluctuations and reduce overfitting. After every BatchNorm, a Dense layer was introduced to better learn data patterns. A GlobalAveragePooling and Dropout layer were placed before the final Softmax. Hyperparameter search provided us with the best fitting values. This layout kept a steady growth before assesting its Val_Accuracy.

B. Local Performance

With a 85-15 dataset split, the model, evaluated with both Early Stopping with patience 30 and ReduceLROnPlateau with patience 5 on val_accuracy, reached the best weight at epoch 73.

- Accuracy: 0.8304
- Val_Accuracy: 0.7021
- Test_Accuracy: 0.6811

This results were found after a fine tuning approach of the model. Its performance on both training and test data lead us to choose this structure for our final submission.

C. Leaderboard Performance

Development Phase Accuracy: 69.53%
Final Phase Accuracy: 70.09%

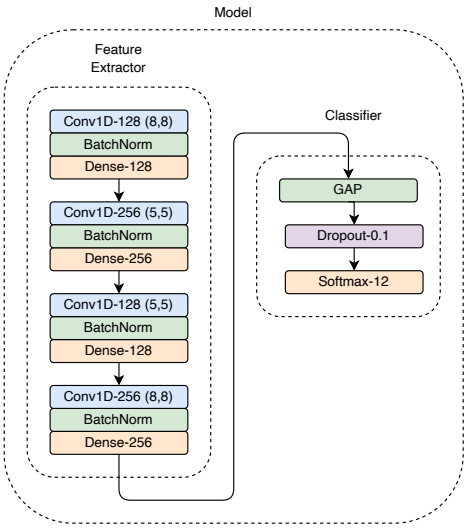

Fig. 6. Structure of final 1DCNN model