AN2DL Challenge 2: Time Series Classification
Artificial Neural Networks and Deep Learning - a.y. 2022/2023
Group: Beck'sPropagation

Davide Franchi
10614321
Codalab: DavideFranchi

Daniele Ferracuti
10580840
Codalab: DanieleFerracuti

Marco Ferrero
10613076
Codalab: marcoferro23

***Abstract*—The purpose of the second AN2DL challenge is to create a model that is able to classify multivariate time series into 12 classes.**

1. Introduction

This challenge consists in implementing a model that can perform multivariate time series classification.

The given dataset is composed of a numpy array of dimension `(num_samples, window_size, channels)`. Window size is fixed at 36 for each sample, and the number of channels is 6, while the number of samples is roughly 2500. The first thing we noticed is that our dataset was strongly unbalanced, with some classes being represented by 30 tuples while others having over 700. To try and alleviate this issue we used class weights during training and some data augmentation techniques described in the next sections.
We divided the given dataset file using the `train_test_split` function from `scikit-learn` with an `80/20` ratio.

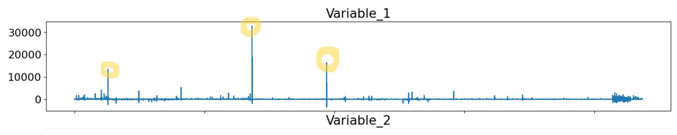

Figure 1. Outliers on the first variable

As we inspected the dataset, we noticed the presence of outliers. This could be a problem if not dealt with correctly. In the next sections, we will explain the methods we employed to attack the problem and results we obtained.

2. Data Processing

2.1. Time series Pre-processing

2.1.1. Normalize on window size. The first thing we tried is perform a z-normalization of the values focusing on the window_size dimension. This turned out to be worse than no normalization at all, because of the presence of outliers in some windows.

2.1.2. Normalize on whole dataset. We then tried to perform a z-normalization, this time focusing on the whole dataset. To perform this, we reshaped the dataset to obtain a two dimensional array of shape `(num_samples * window_size, channels)` and then performed the normalization on all the values. This method was better than the window-normalization.

2.1.3. Change window size with linear interpolation. In order to explore the effect of different window sizes on the performance of our models, we utilized the `Resize` class from the `tsaug` library to resample our time series data to different frequencies. Specifically, we resampled the data to a larger frequency, resulting in a larger window size, and used linear interpolation to generate new sequences with the same shape as the original ones. However, after comparing the model performance on the resampled data with the original data, we found that using the original window size resulted in slightly better performance. As a result, we chose to use the original data with its original window size for all of our subsequent training runs.

2.1.4. Removing and adding variables. We noticed that the six channels of the dataset had a quite similar signal shape, some of them even the same value range. We experimented with removing one or more variables, or adding an artificial one with the average values of other variables. We decided to not perform this kind of operation on our best models, because we did not see any improvements.

2.2. Data Augmentation

Using data augmentation techniques has been an effective way to increase the size of our dataset and improve the generalization of the models. In our study, we implemented a custom Keras Sequence class to generate sequence batches for training and used the tsaug library to apply various augmentations to the data, including Noise Injection, Crop, TimeWarp, Reverse, Drift, and others. Additionally, we created a custom function to shift sequences in time with a random number of time samples.

After evaluating the impact of these augmentations on model performance, we found that the Noise Injection and sequence shifting transformations were the most effective, resulting in significant improvements in performance. Therefore, we decided to use these two augmentations with the optimal parameters for all of our training runs.

Augmentation parameters	
AddNoise	range = (0.01-0.08)
image_shifting	random values

3. RNN basic models

Our first approach to the challenge was to derive as many models as possible and reason about which one might be the best to focus on. In particular, we initially created very simple RNN modes, using basic LSTM or (better) BiLSTM layers, and then,

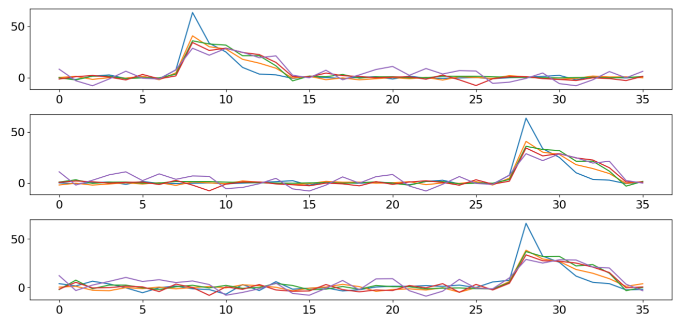
Figure 2. Original, shifted and noise added augmentation.

exploiting more-complex structures, implementing techniques such as Attention mechanism and transformers to obtain more accurate results.

We tried using different types of loss functions (such as CategoricalCrossentropy and CategoricalFocalLoss) and optimizers (such as Adam, LazyAdam and RMSprop), but in all the models analyzed we found the following to be the most suitable:

- **Loss function** : CategoricalCrossentropy
- **Optimizer** : Adam

3.1. GRU model

We also attempted to use a Gated Recurrent Unit (GRU) model as an alternative to the BiLSTM model. However, the results were not as satisfactory as those obtained with the BiLSTM model, with an accuracy of only 0.53498. This may be due to the fact that the GRU model does not have an explicit memory component, unlike the BiLSTM model which has both forward and backward LSTM cells that can retain information from previous timesteps. Additionally, the GRU model has fewer parameters than the BiLSTM model, which may result in a lower capacity to learn complex patterns in the data. Further investigation is needed to determine the optimal architecture for our task.

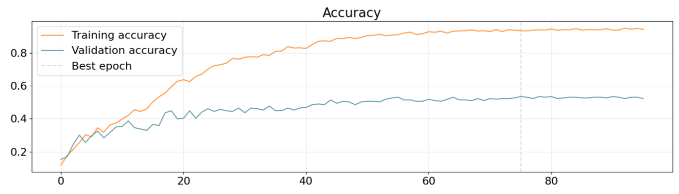

Figure 3. Basic GRU training run

3.2. BiLSTM model

Of all the baseline models we tried, BiLSTM was the most efficient and we used it to test the other components of the neural network such as preprocessing and data augmentation. We found that using 256 layers for the BiLSTM model resulted in the best performance, with an accuracy of 0.748. BiLSTM was the most efficient model for our task and we used it to evaluate the impact of other components on the performance of the model.

4. GTN

Gated Transformers Network is a model that is able to perform multivariate time series classification with performance

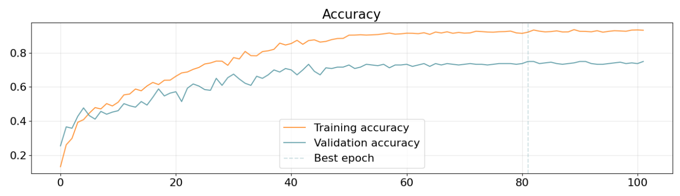

Figure 4. Basic BiLSTM training run

comparable to state of the art methods. [1]
We implemented the model in Tensorflow, but the performance were worse than expected (ranging between 0.20 and 0.40 validation accuracy). The model was struggling to learn, as weights updated stochastically. A possible explaination for this misbehaviour could be the very small window size of our dataset. Transformer-based models perform better when the time step is considerably large.

5. ResNet Based Models

We decided to implement ResNet-based models to implement more complex architecture based on the Residual blocks.

A residual block is used to enable the network to learn residuals, or the difference between the input and the desired output, instead of the desired output itself. This helps to alleviate the vanishing gradient problem, which is a common issue in deep neural networks.

The structure of a residual block in a ResNet typically consists of two or more convolutional layers, with the addition of a shortcut connection. The shortcut connection is used to bypass one or more of the layers in the block and directly add the input to the output of the block. This allows the network to learn residuals instead of the desired output.

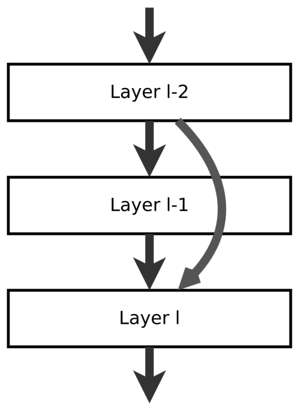

Figure 5. Residual block

5.1. 1D Conv ResNet

In our initial approach, we implemented a Residual Neural Network (ResNet) with three residual blocks and 1D convolution layers. The number of feature maps increased as we progressed deeper into the network. We utilized batch normalization, dropout, and the ReLu activation function to improve the accuracy of the blocks. However, the results obtained with this model were significantly worse than those achieved with a basic BiLSTM model. These findings suggest that the ResNet architecture with 1D convolution may not be well-suited for this particular problem and that alternative approaches should be explored.

5.2. BiLSTM ResNet

In an effort to improve upon the basic BiLSTM model, we implemented a ResNet-based model with recurrent layers. This approach proved to be more effective than the basic model, resulting in an approximately 3-4 percent increase in validation accuracy.

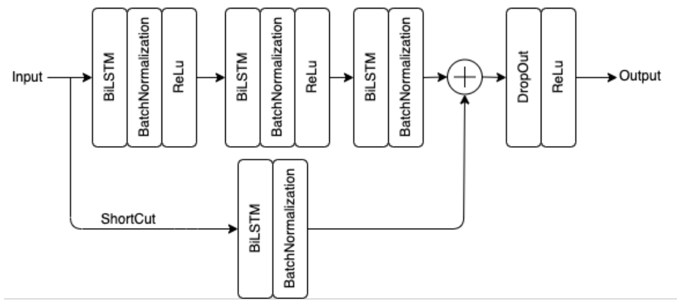

Figure 6. ResNet block with BiLSTM layers

The optimal structure of this network was found to consist of two residual blocks as shown in Figure 6, with the first block containing 128 units and the second block containing 256 units. Dropout layers with a rate of 0.5 were placed at the end of each residual block, and a dropout layer with a rate of 0.3 was included in the final classifier block. The model had a total of more than 6 million parameters.

While adding additional blocks to this model did not result in further performance improvements, it is likely that a large number of trainable parameters made it difficult to achieve good training results. The final model achieved a validation accuracy of 0.79.

5.3. BiLSTM ResNet + Attention Mechanism

Another optimization of the previous model we decided to implement is the attention mechanism. The attention mechanism is a type of layer that allows a model to focus on specific parts of an input when making predictions.

In our specific case it is implemented as a dense layer with a single unit and sigmoid activation, followed by a softmax activation function that scales the weights to sum to 1. These weights are then used to weight the output of the BiLSTM, which is concatenated with the final hidden state of the BiLSTM and used as the input to the classifier block.

This allows the attention mechanism to influence the overall prediction made by the model.

Unfotunerly we didn't see any significant change using this technique, at the same time the best result reached didn't change at all, so we concluded that this is our best model for the classification problem.

6. Training Parameters

6.0.1. Learning rate. We tried using a scheduler, which decreased the learning rate by an exponential factor. We then used ReduceOnPlateau keras function to decrese LR when loss validation stagnated for more then 5 epochs around the same value.

6.0.2. Early Stopping. To have a more accurate result in training we used 200 epochs with early stopping monitoring validation loss with 10 epochs patience.

6.0.3. Callbacks. Using checkpointer we have saved the weights of our model considering its best validation accuracy during the training session

7. Final Model

The final model submitted is the one described in the BiLSTM ResNet + Attention Mechanism subsection.

It reached 0.78 in validation accuracy but a lower score on Codalab platform (0.69). Here you can see the confusion matrix of this model predictions:

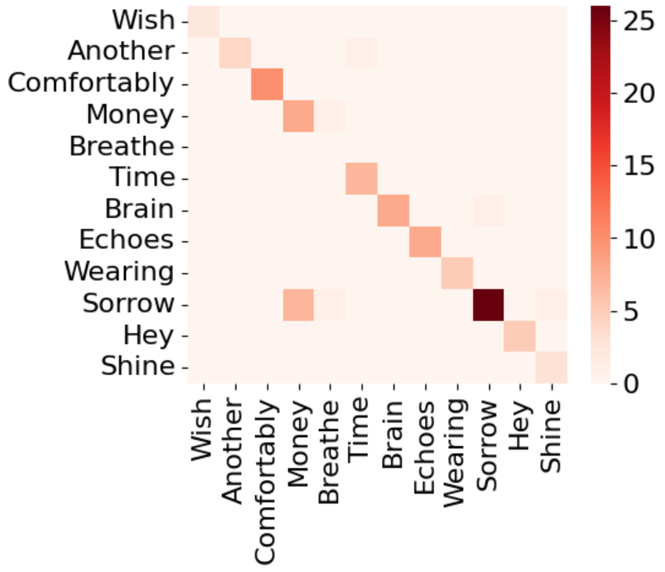

Figure 7. Confusion matrix for a small test set

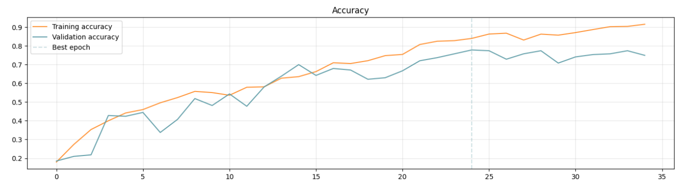

Figure 8. Training run for BiLSTM ResNet + Attention

8. Possible Optimizations

Some possible optimization that we could have done to our best model are:

- **More work on preprocessing**: The dataset provided was very challenging, and we could not infer which preprocessing method was optimtal simply by looking at the samples. We are sure that with more tinkering we could have found a suitable preprocessing pipeline to boost the classification results.
- **Ensemble**: Ensembling different weak models gives a more robust model. We did something similar with our gating experiments (GTN, GRU, Attention), but did not implement a strategy for weighting the models' predictions to do a proper ensemble.

References

[1] Minghao Liu, Shengqi Ren, Siyuan Ma, Jiahui Jiao, Yizhou Chen, Zhiguang Wang, and Wei Song. Gated transformer networks for multivariate time series classification, 2021.