Multivariate time series multiclass classification

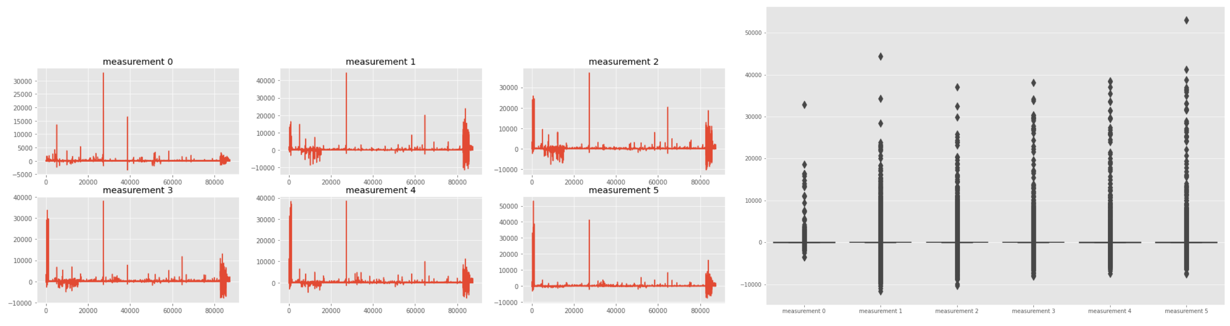
Calligaro Nicola, Castiglia Danilo, Cimpeanu Vlad

1 Introduction

In this task, we are required to correctly classify samples in the multivariate time series format. In other words, 6 different time series are provided and the goal is to classify them with a single label among 12 possible labels.

2 Data preprocessing and data augmentation

Since no information about the data has been provided, first we inspect the given time series and try to extract some useful insights that may be used for preprocessing.
As we can see from Figure 1, there are several outliers for each measure, furthermore, those outliers are incredibly distant from the majority of the data, as matter of fact, for each measure, the median is in the order of 10, whereas most of the outliers are 3 orders of magnitude higher [1].
Because we do not know anything about our dataset, we cannot use our domain knowledge to handle the outliers, thus we cannot consider removing them, as they may not be errors in the measurements, but true data points.
Due to the presence of the outliers, we cannot use MaxMin scaling or z-score standardization, since they rely on statistics that are sensible to the outliers, thus we decide to scale each measurement applying the Robust scaling, indeed it depends on the median and the interquantile range that are not influenced by the outliers.



(a) (b)

Figure 1: (a) Visualization of the six measures. (b) Boxplot for each measurement.

Then, we inspect the distribution of the classes and we notice the dataset is highly unbalanced, and some classes such as 0, 4, 7, and 11 have few samples. Facing up with this issue, first, we try introducing a set of weights to be applied when computing the loss function to make ”weigh” more wrong predictions on classes that have high weight values. In this way, we force the model to learn more carefully how to recognize the most challenging classes.
Moreover, we try to directly optimize the macro f1-score using the soft f1-loss function found here, unfortunately, this loss function provided good results only on few experiments.
Finally, since there are only few samples, we try to augment the dataset. As first attempt, we add some Gaussian noise at run-time using the `GaussianDropout` layer after the `Input` layer with a rate of 0.05. We choose `GaussianDropout` over `GaussianNoise` since the former controls the noise’s standard deviation according to the amplitude of the input, whereas the latter has a static standard deviation.

[1] This is clear also by looking at the boxplots, indeed whiskers almost collapse on the same line.

Last but not least, we make a strong assumption about the data that may not be true: we assume the windows belonging to the same class are sequential, thus, we can unroll all the windows and create new samples using a sliding window with 36 size and stride 1.
Using this specific augmentation we get way better results on the validation set, so we have empirical evidence this augmentation works, even though, we are aware this approach may lead to completely wrong results for different datasets. A reason why this method works is we are shifting each original sample and substituting missing values with other values coming from the same class[2].
When we divide the dataset into training set and validation set we are careful not to have any repetition of the same timestamp in the two sets (training and validation). This needs to be done in order to avoid information leak of the validation set.

3 Naive model

We start with an LSTM-based model, by stacking 2 bidirectional `LSTM` layers with 64 neurons and a `Dense` layer with `softmax` activation achieving 70% of accuracy. Since the model suffers from overfitting, we add `Dropout` with 0.5 rate, without any improvement. We notice the training accurcacy reaches 99% after few epochs, therefore we try also to reduce the learning rate from 10^{-3} to 10^{-4} but it produced only worse results. As the model seems to be too complex, given the data and the task, we try only one bidirectional `LSTM` layer with 64 neurons, achieving 66% accuracy. By reducing the complexity of the model, we cannot increase the generalization of the model, instead, we are getting only poorer results, hence we try to increase the model complexity even though the model is already overfitting.
The best result we achieve is 71% accuracy using 2 bidirectional `LSTM` with 128 neurons and `Dropout` with 0.7 rate.
Then, we try another approach, by using a model with the following layers: `Conv1D`, `MaxPooling1D`, `Conv1D`, `GlobalAveragePooling1D`, `Dropout`, `Dense` with `relu` activation and finally a `Dense` with `softmax` activation. The best model with this configuration has `Conv1Ds` layers with 256 filters and kernel size 3, 128 neurons in the fully connected hidden layer and 0.5 dropout rate, reaching 74.5% accuracy. This architecture is the only one for which the soft f1 loss function provides better results, indeed increasing the number of filters to 512 we reach 75.1% of accuracy. We conclude convolution-based models seems to be more promising.

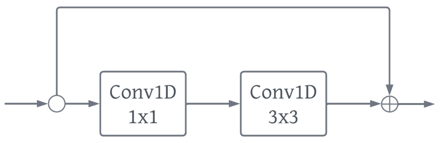
4 Resnet1D

In this section, we take inspiration from ResNet architecture using Residual blocks as the main components of the model. We build a residual block using full pre-activation as illustrated here, thus, before every `Conv1D` layer we apply a `BatchNormalization` layer and `relu` activation layer.
`BatchNormalization` layers are strongly affected by the outliers, to mitigate their effect we set BatchNormalizations' momentum to 0.
We start using a model with a `Conv1D` layer with 64 filters and kernel size 3, `MaxPooling1D`, 3 `ResidualBlocks` by 64 filters, `GlobalAveragePooling1D`, `Dense` with `softmax` activation.
The first model reaches 68% accuracy and we decide to add new `ResidualBlocks` until no improvement can be seen. The best result achieved is by using 3 `ResidualBlocks` by 64 filters, `MaxPooling` and 4 `ResidualBlocks` by 128 filters getting 70% of accuracy.
Finally, we try to implement an alternative version of the residual block, as shown in Figure 2(a), where we try to sum also over the channels through the skip connections. By stacking 3 new `ResidualBlocks` by 64 filters and 2 new `ResidualBlocks` by 128 filters we achieve 72% of accuracy.
These results seem disappointing considering that much simpler models got us better performances.

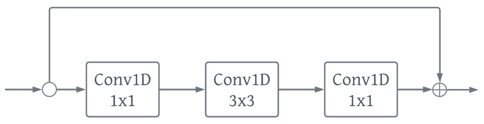
5 Multihead models

Finally, we implement a multi-head architecture, more precisely, we build a model such that each channel of the input (in this case each time series) is redirected to a specific stack of layers as follows: `Conv1D`, `MaxPooling1D`, `Conv1D` where each convolution layer has 128 filters with a kernel size of 3. The stacks'

[2]We can find an analogy in CutMix augmentation for images



(a)



(b)

Figure 2: (a) Revisited residual block. (b) Residual 1D block

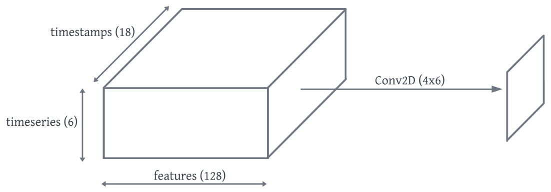


Figure 3: 2D convolution on extracted features.

outputs are concatenated and reach the output layer passing through a `GlobalAveragePooling1D`, `Dropout` with rate of 0.2 and `Dense` of 128 neurons and `relu` activation.
Due to a distraction error, during our first attempt we forgot to connect each channel to a different stack, instead, we connected the entire input to all the stacks ending with a model which does not seem to have any sense. Nevertheless, this 'bugged' model reached 74.4% accuracy in the validation set and 74.5% in the test set[3].
Afterward, we fix the error previously made and correctly implement its original idea. However, the performances of the resulting model are not as good as the ones displayed by the 'bugged' implementation, reaching 68% of accuracy on the validation set (adjusting rate in `Dropout` layer at 0.1). Comparing the two versions of multi-head models just mentioned and analyzing the results obtained, we can state that the convolution part (set of heads) is able to output a more valuable result when taking as input the entire set of time series rather than trying to extract good features considering in input only a single time series per head and then concatenate these for feeding the fully connected part of the network.
As last try concerning this approach, we add a `Conv2D` layer with 128 filters of size 4x6 after the concatenation of results of all heads: these now are actually stacked on a new dimension instead of concatenated on an already existing dimension, as shown in Figure 3(the `GlobalAveragePooling1D` layer has been changed with `GlobalAveragePooling2D` layer too). In this way, we apply the technique used usually when dealing with convolution in images and make the channel's dimension the one regarding features generated in the last convolution layer present in the heads. This approach combines the idea of extracting features by looking at single time series and then at the entire set of these (since we set filter size equal to 6 on the time series axis). It reaches 74.9% of accuracy on validation set.
Next we try to build an architecture with 3 heads, and the idea is that one head process the input as it is, while the second head process the input cropped of 1 timestamp from the start of the window, whereas the last head process the input cropped of 1 timestamp from the end of the window. To obtain this result we used a `Cropping1D` layer followed by a `ZeroPadding1D` layer.
The complete architecture is the following: one of the heads is composed of `Conv1D`, `MaxPooling1D`, `Conv1D`; while the other two are composed of `Cropping1D`, `ZeroPadding1D`, `Conv1D`, `MaxPooling1D`, `Conv1D`. Each convolution layer has 128 filters with a kernel size of 3. The stacks' outputs are concatenated and reach the output layer passing through a `GlobalAveragePooling1D`, `Dropout` with rate of 0.2 and `Dense` of 128 neurons and `relu` activation.
This setup gave us very encouraging results, performing a 76% accuracy in the validation set but unfortunately on the test set was not performing as good and we only achieved 72.6% accuracy.
We tried also to change the number and size of the filters and also experimented cropping instead of 1 timestamp, 2, 3, 5 and 10 timestamps but the accuracy was the same or even worst as before.

[3]We submitted this model to the second phase of the challenge since it was the one with better results in the first phase.