# Time Series Classification

# Oscar Aragon, Andrea Borsatto, Riccardo Donati

1. **Introduction**

The scope of the project was to correctly classify samples, belonging to 12 different classes, in the multivariate time series format. We were given a dataset in .npy format containing the time series of shape (2429, 36, 6). Our goal was to design the classification network and apply the correct preprocessing techniques to our data.

1. **Dataset description**

The time series was already segmented in windows of 36-time units and each time series had 6 features per timestamp. The entire dataset was given to us as a numpy file called X\_train.npy, while the target classes to classify in a separate file called y\_train.npy.

* 1. **Class imbalance**

The dataset we were given was highly imbalanced, for this reason we tried different techniques to balance it:

* + - **Downsampling**: The scope of this technique was to reduce the number of samples of the majority class that created misclassifications since they were located on the decision boundary between two classes. Our first try was with the Tomek Links Rule, which finds pairs of samples belonging to different classes that have the smallest Euclidean distance in the feature space and deletes the sample of the majority class. This approach didn’t reduce the size of the majority class enough to produce meaningful results.

After that we tried the Edited Nearest Neighbors Rule that classifies each sample with a k=3 Nearest Neighbors. If a sample belonging to the majority class is misclassified by its Three-Nearest-Neighbors, it is deleted from the dataset. This technique was more effective in reducing the size of the majority class but did not produce interesting results.

* + - **Upsampling**: The scope of this technique was to increase the number of samples in the minority classes to have a uniform number of instances. We tried simple random duplication, but it generated a lot of overfitting, so we applied multiple types of augmentation to each sample, as described in the “Data augmentation” paragraph.

We also tried a technique to generate synthetic examples called SMOTE (Synthetic Minority Oversampling Technique) that selects a minority class instance “a” at random and finds its k nearest minority class neighbors. Then one of the k nearest neighbors “b” is chosen and a line segment connecting “a” and “b” is created in the feature space. The synthetic sample is generated on that line. This technique was very promising at the beginning, but the synthetic samples were too much different from the real ones and they did not produce any performance improvement, so we had to discard this technique.

* 1. **Outliers Detection**

Inspecting the time series, we realized that every feature had some samples with a very large range of values inside the window influencing the overall data statistics. This means that outliers were present and we needed to detect and remove them. We decided to use one of the most robust methods, called Median Absolute Deviation (MAD), an alternative to standard deviation to find outliers in one-dimensional data. It is defined as the median of the absolute deviations from the series median, and its values are more robust than standard deviation with respect to the range of the outliers. As a result, the limits for determining outliers using MAD are less influenced by the range of outliers (as it happens using the standard deviation, for example).

This method, unfortunately, didn’t lead to significant results in terms of accuracy, since the detected number of outliers in the dataset was significant and removing them degraded the quality of our data. We ended up using different normalization approaches.

* 1. **Data normalization**

One of the most important phases of the preprocessing step is the normalization. Its goal is to transform features to be on a similar scale. This improves the performance and training stability of the model.

We started using the MinMaxScaler, which scales the features to a given range ([0,1] in our case) and the StandardScaler, which standardizes features by removing the mean and scaling to unit variance. These scalers are very sensitive to outliers and this resulted in poor performance. We then moved on using more robust scalers:

* + - RobustScaler, this Scaler removes the median and scales the data according to the quantile range (25,75)
    - PowerTransformer, a family of parametric, monotonic transformations that are applied to make data more Gaussian-like.

We got improvements with both approaches and ended up using the second one.

* 1. **Data augmentation**

Another approach to address the class imbalance problem was to use augmentation, the goal was to expand our dataset and at the same time regularize our models

* + - **GAN:**

We tried a generative approach for class balancing using a Generative Adversarial Network.

Generative modeling is an unsupervised learning task in machine learning that involves automatically discovering and learning the regularities or patterns in input data in such a way that the model can be used to generate or output new examples that plausibly could have been drawn from the original dataset.

GANs are a clever way of training a generative model by framing the problem as a supervised learning problem with two sub-models: the generator model that we train to generate new examples, and the discriminator model that tries to classify examples as either real (from the domain) or fake (generated). The two models are trained together in an adversarial zero-sum game.

We used the DoppelGANger library, which is specific for the generation of time series.  
This method was very promising, but it didn't give the desired results plausibly for a scarcity of data.

* + - **Classic data augmentation:**

We selected a subset of suitable data augmentation techniques for time series from the tsaug library:

* + - * **Time Warp**, Random changing of the speed of timeline.
      * **Reverse**, Reverse the time series.
      * **Drift**, Drifts the value of time series from its original values randomly and smoothly.
      * **Add Noise**, Add random noise to every time point.
      * **Convolve**, Convolve time series with a kernel window.

We applied them 10 times to the original training dataset, concatenating the new augmented samples each time, selecting a random subsample of the augmentations

1. **Model development**

We implemented different models and we compared their performance on our augmented dataset:

* + - We first implemented a simple LSTM network which is capable of learning long-term dependencies. Our model has two LSTM layers with dropouts to reduce overfitting.
    - Bidirectional LSTM to extract information from both directions and to exploit sequential dependencies of the input sequence. This model gave us better results than the plain LSTM model.
    - A simple ResNet with 3 Residual Blocks to exploit their skipped connections.
    - Transformer with the same architecture as the one in the paper “*Attention Is All You Need”* and applied to time series instead of natural language. It did not give us good results but it was an interesting method of applying the attention mechanism to time series classification.
    - 1D Convolutional Neural Networks with multiple heads, where each head of the model reads the time steps of the input using a different sized kernel. We used a three-headed model with three different kernel sizes of 3, 5, 8, allowing the model to read and interpret the sequence data at three different resolutions. The interpretations from all three heads are then concatenated within the model and interpreted by a fully-connected layer before a prediction is made.

This model was the best performing one among those mentioned above.

1. **Model training**

The model that gave us a higher level of accuracy was the multihead 1D convolutional, so we tried, starting from that, to create an ensemble of models that would allow us to improve our predictive ability.

After various tests we obtained our final ensemble by joining 6 basic multihead models trained on the training set normalized with the PowerTransformer and then augmented, as specified in the previous paragraph.

We used six different batch sizes: 4, 8, 16, 32, 64, 128 and performed a weighted average of the predictions, using the same weight for all the models because their performances were similar. We obtained in this way an improvement in the overall accuracy on the test set.

1. **Conclusions**

We tried many approaches and techniques, some were unsuccessful, but they allowed us to get a deeper understanding of our dataset which allowed us to reach a good final accuracy on the test set of 74.44%.

Possible improvements could be made with ensembles of structural different networks and with even more focus on the balancing of the dataset, exploring other upsampling and downsampling strategies and trying more complex techniques for the generation of synthetic samples.