# Kaggle - Time Series Classification

## Universidad Carlos III de Madrid

Duarte Moura

Roberto Martínez - Guisasola Guerrero

December 19, 2024

*This report documents the implementation and evaluation of the model we used for time series classification. The pipeline involves preprocessing, augmentation, feature engineering, and model training.*

## 1 Introduction

Time Series Classification (TSC) involves categorizing sequential data into predefined categories. This project presents a deep learning pipeline designed to address key challenges associated with TSC, including:

- Handling imbalanced datasets.
- Extracting meaningful features from raw data.
- Effectively modeling temporal dependencies.

This report outlines each stage of the pipeline, from preprocessing to model evaluation, and discusses the results and future improvements.

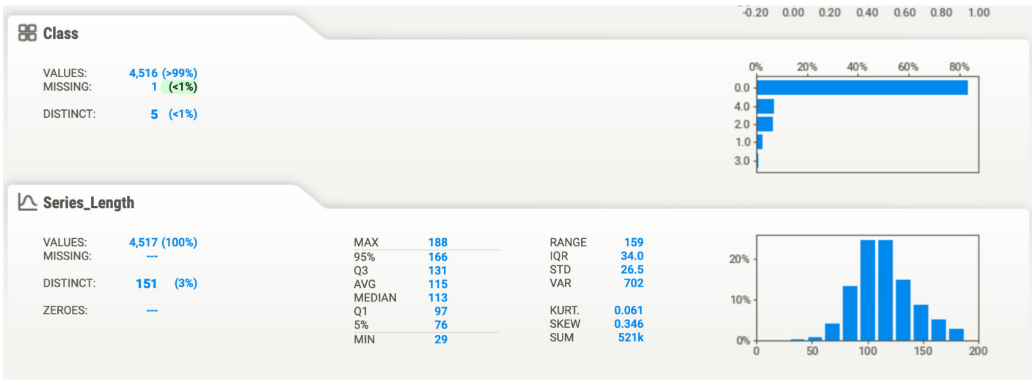
## 2 Pre-processing

### 2.1 Data Description

The dataset comprises time series with 187 features per sample:

- **Training Set:** 87,554 samples.
- **Test Set:** 21,892 samples.
- **Classes:** 5 categories with significant class imbalance.

To gain initial insights, a Sweetviz report was generated, highlighting the data distribution and class proportions.



**Class**

VALUES: 4,516 (>99%)
MISSING: 1 (<1%)
DISTINCT: 5 (<1%)

**Series_Length**

VALUES: 4,517 (100%)
MISSING: ---
DISTINCT: 151 (3%)
ZEROES: ---

| | |
|---|---|
| MAX | 188 |
| 95% | 166 |
| Q3 | 131 |
| AVG | 115 |
| MEDIAN | 113 |
| Q1 | 97 |
| 5% | 76 |
| MIN | 29 |

| | |
|---|---|
| RANGE | 159 |
| IQR | 34.0 |
| STD | 26.5 |
| VAR | 702 |
| KURT. | 0.061 |
| SKEW | 0.346 |
| SUM | 521k |

Figure 1: Data Distribution

### 2.2 Handling Missing Values

Missing values were replaced with `NaN` and subsequently interpolated using a linear method to maintain temporal continuity. This ensured the integrity of the sequences for downstream processing while minimizing information loss.

### 2.3 Feature Engineering

We computed rolling statistics using a sliding window of size 3:

- **Rolling Mean:** Captures the local average trend.
- **Rolling Standard Deviation:** Captures local variability.

Each sample was transformed into a 3-dimensional representation: `(original, mean, std)`. The resulting dataset shape per sample was `(187, 3)`. By including these additional features, the LSTM model gained richer contextual information at each time step.

### 2.4 Data Balancing

Class imbalance was addressed using the `tsaug` library to generate synthetic time-series data. This approach was preferred over alternatives like SMOTE, which struggled with the temporal nature of the data.

### 2.5 Class Distribution

Original Class Distribution:

- **Class 0:** 72,471 samples
- **Class 1:** 2,223 samples
- **Class 2:** 5,788 samples
- **Class 3:** 641 samples
- **Class 4:** 643 samples

Using augmentation, we either:

- Matched all classes to the majority class (Class 0).
- Balanced all classes to 80,000 samples each.

## 3 Model Architecture

### 3.1 LSTM with Masking and Attention

The model architecture leverages an LSTM with masking and attention mechanisms to classify time-series data effectively:

- **Input:** 187 time steps with 3 features each.
- **LSTM Layer:** Bidirectional with dropout regularization to capture temporal dependencies.
- **Attention Mechanism:** Computes attention weights to focus on the most relevant time steps, generating a context vector for classification.
- **Output:** Fully connected layers for classification into 5 categories.

### 3.2 Masking and Padding

To handle variable-length sequences:

- **Padding:** Shorter sequences were padded to match the longest sequence in a batch, enabling efficient batch processing.
- **Masking:** A binary mask identified valid time steps, ignoring padded values during computations.

During attention computation, padded positions were assigned `-inf`, ensuring their weights became zero after applying softmax. This strategy ensured the model focused solely on meaningful parts of the sequence.

### 3.3 Training and Validation

Key hyperparameters for the LSTM model:

- **Hidden Dimensions:** 128
- **Layers:** 2
- **Dropout:** 0.5
- **Learning Rate:** 0.001
- **Patience for Early Stopping:** 6 epochs

### 3.4 Results

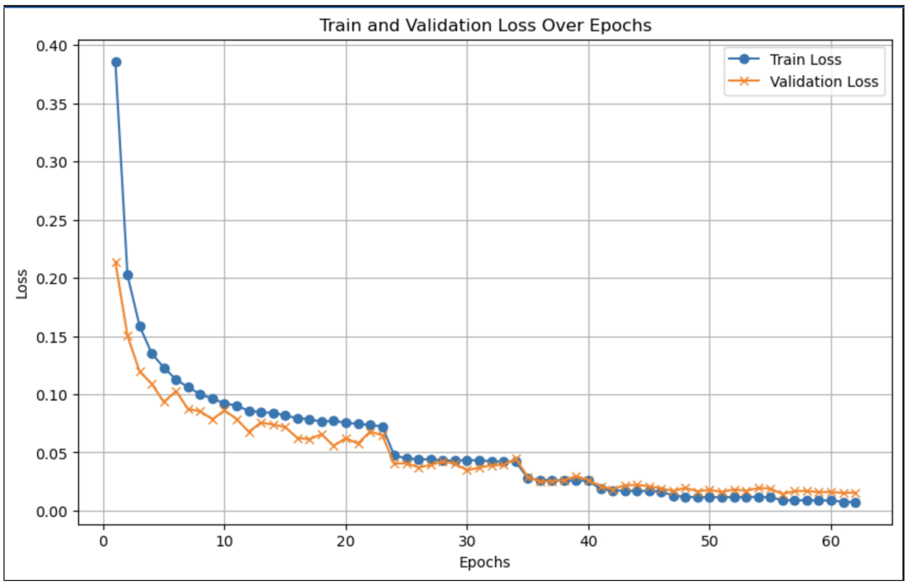


Figure 2: Test and Validation Loss over Epochs

The model achieved high accuracy but exhibited some overfitting, as indicated by the divergence between training and validation loss. We tried to reduce model complexity (like number of layers or hidden dimensions) to mitigate this issue, but it resulted in worse performance in the end.

### 3.5 Other Methods Explored

In addition to the LSTM, we explored several other methods to improve classification performance. These alternatives, however, did not outperform the LSTM:

- **CNN-LSTM Hybrid:** Adding a CNN layer before the LSTM failed to capture meaningful patterns, even with an attention mechanism between both layers.
- **Class-Weighted LSTM:** In this method we just left the classes with the same number of data points to keep the weights unbalanced and used them in the loss function. Didn't work very well compared to the augmented LSTM.
- **Inception Time:**[1] A pre-developed time-series classification model. Despite its reputation (we saw good reviews), it performed poorly.
- **Feed-Forward Neural Network:** A fully connected neural network yielded the worst performance, lower than 0.5, so it was discarded.

## 4 Conclusion

Our proposed pipeline successfully addressed the challenges of time-series classification, delivering strong performance despite class imbalance and complex temporal dependencies. Future work could explore:

- **Autoencoders:** For dimensionality reduction while preserving temporal patterns.
- **Variational Autoencoders (VAEs):** To generate realistic synthetic data for minority classes.

These techniques could further enhance the model's capabilities and robustness.

[1]https://github.com/flaviagiammarino/inception-time-pytorch

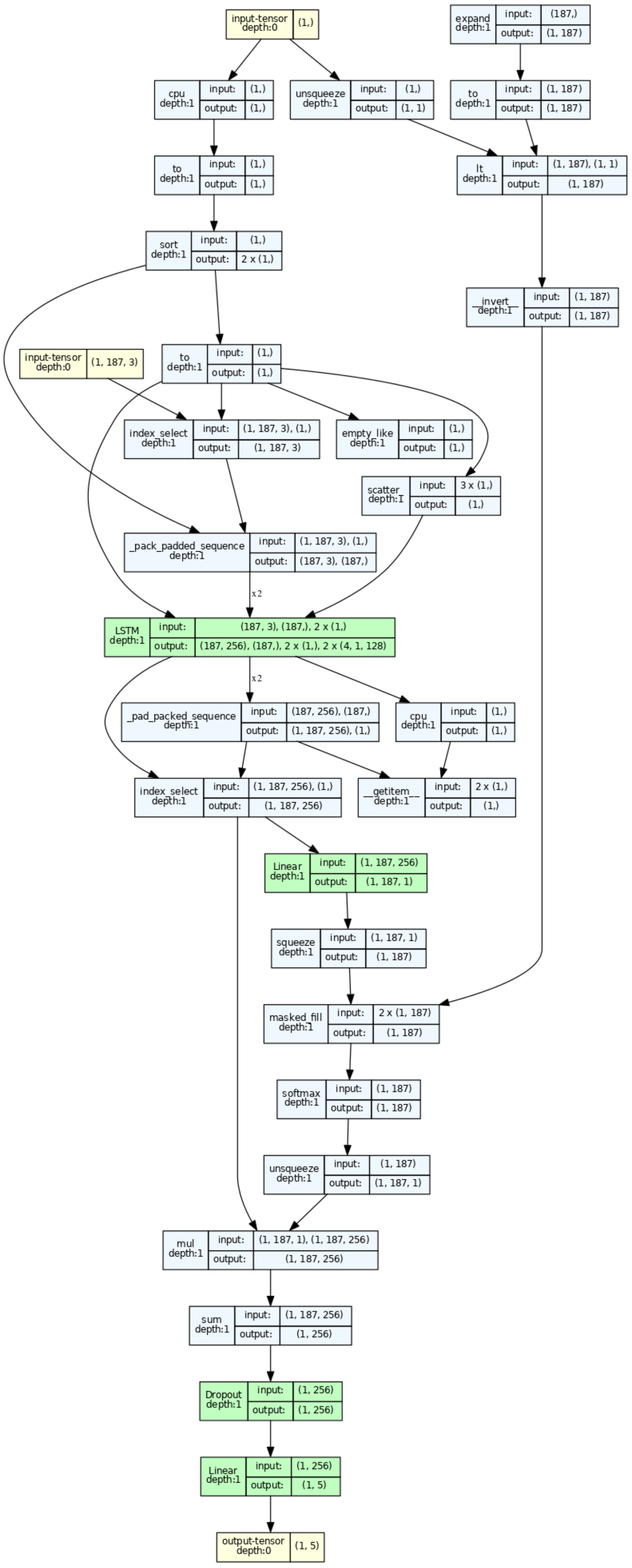

input-tensor depth:0 (1,)
cpu depth:1 input: (1,) output: (1,)
unsqueeze depth:1 input: (1,) output: (1, 1)
expand depth:1 input: (187,) output: (1, 187)
to depth:1 input: (1, 187) output: (1, 187)
to depth:1 input: (1,) output: (1,)
lt depth:1 input: (1, 187), (1, 1) output: (1, 187)
sort depth:1 input: (1,) output: 2 x (1,)
__invert__ depth:1 input: (1, 187) output: (1, 187)
input-tensor depth:0 (1, 187, 3)
to depth:1 input: (1,) output: (1,)
index_select depth:1 input: (1, 187, 3), (1,) output: (1, 187, 3)
empty_like depth:1 input: (1,) output: (1,)
scatter depth:1 input: 3 x (1,) output: (1,)
_pack_padded_sequence depth:1 input: (1, 187, 3), (1,) output: (187, 3), (187,)
x2
LSTM depth:1 input: (187, 3), (187,), 2 x (1,) output: (187, 256), (187,), 2 x (1,), 2 x (4, 1, 128)
x2
_pad_packed_sequence depth:1 input: (187, 256), (187,) output: (1, 187, 256), (1,)
cpu depth:1 input: (1,) output: (1,)
index_select depth:1 input: (1, 187, 256), (1,) output: (1, 187, 256)
__getitem__ depth:1 input: 2 x (1,) output: (1,)
Linear depth:1 input: (1, 187, 256) output: (1, 187, 1)
squeeze depth:1 input: (1, 187, 1) output: (1, 187)
masked_fill depth:1 input: 2 x (1, 187) output: (1, 187)
softmax depth:1 input: (1, 187) output: (1, 187)
unsqueeze depth:1 input: (1, 187) output: (1, 187, 1)
mul depth:1 input: (1, 187, 1), (1, 187, 256) output: (1, 187, 256)
sum depth:1 input: (1, 187, 256) output: (1, 256)
Dropout depth:1 input: (1, 256) output: (1, 256)
Linear depth:1 input: (1, 256) output: (1, 5)
output-tensor depth:0 (1, 5)

Figure 3: Full Model Architecture