main

May 19, 2025

1 Índice

Hay imágenes en las celdas markdown, es mejor mirar el notebook desde `repo/src/main.ipynb`.

2 Members

- Alejandro Cortijo Benito
- Alejandro García Mota

3 Librerías

[ ]:
```
import warnings

warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=UserWarning)

import pandas as pd
import plotly.graph_objects as go
import matplotlib.pyplot as plt

from sklearn.ensemble import IsolationForest

import numpy as np
import pandas as pd
from tqdm import tqdm
```

```
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, RepeatVector, TimeDistributed,␣
↪Dropout, Bidirectional, Attention, LayerNormalization, Concatenate, Conv1D,␣
↪Conv2D, Reshape, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import Sequence
from tensorflow.keras.regularizers import l2
from tensorflow.keras.models import Model

import pywt

from scipy.ndimage import label
```

```
2025-05-18 08:31:12.022909: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:467] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
E0000 00:00:1747549872.137418    3320 cuda_dnn.cc:8579] Unable to register cuDNN
factory: Attempting to register factory for plugin cuDNN when one has already
been registered
E0000 00:00:1747549872.171711    3320 cuda_blas.cc:1407] Unable to register
cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has
already been registered
W0000 00:00:1747549872.428844    3320 computation_placer.cc:177] computation
placer already registered. Please check linkage and avoid linking the same
target more than once.
W0000 00:00:1747549872.428876    3320 computation_placer.cc:177] computation
placer already registered. Please check linkage and avoid linking the same
target more than once.
W0000 00:00:1747549872.428877    3320 computation_placer.cc:177] computation
placer already registered. Please check linkage and avoid linking the same
target more than once.
W0000 00:00:1747549872.428879    3320 computation_placer.cc:177] computation
placer already registered. Please check linkage and avoid linking the same
target more than once.
2025-05-18 08:31:12.456468: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
```

4 EDA

[ ]:
```python
train = pd.read_parquet('../data/train.parquet')
test = pd.read_parquet('../data/test.parquet')

def eda_summary(df, name):
    print(f"--- {name} ---")
    print("Shape:", df.shape)
    print("\nColumnas y tipos de datos:")
    print(df.dtypes)
    print("\nPrimeras filas:")
    print(df.head())
    print("\nValores nulos por columna:")
    print(df.isnull().sum())
    print("\nEstadísticas descriptivas:")
    print(df.describe(include='all'))
    print("\n")

eda_summary(train, "Train Dataset")
```

```
--- Train Dataset ---
Shape: (14728321, 89)

Columnas y tipos de datos:
id                    int64
channel_1           float32
channel_10          float32
channel_11          float32
channel_12          float32
                     …
telecommand_376     float64
telecommand_38      float64
telecommand_39      float64
telecommand_40      float64
is_anomaly            uint8
Length: 89, dtype: object

Primeras filas:
   id  channel_1  channel_10  channel_11  channel_12  channel_13  channel_14  \
0   0    0.13791         0.0         0.0    0.317175    0.371764    0.297205
1   1    0.13791         0.0         0.0    0.317175    0.371764    0.297205
2   2    0.13791         0.0         0.0    0.317175    0.371764    0.297205
3   3    0.13791         0.0         0.0    0.317175    0.371764    0.297205
4   4    0.13791         0.0         0.0    0.317175    0.371764    0.297205

   channel_15  channel_16  channel_17  …  telecommand_351  telecommand_352  \
0    0.130113    0.766769    0.349474  …             0.0              0.0
1    0.130113    0.766769    0.349474  …             0.0              0.0
```

```
2    0.130113    0.766769    0.349474  …             0.0             0.0
3    0.130113    0.766769    0.349474  …             0.0             0.0
4    0.130113    0.766769    0.349474  …             0.0             0.0

   telecommand_353  telecommand_354  telecommand_36  telecommand_376  \
0              0.0              0.0             0.0              0.0
1              0.0              0.0             0.0              0.0
2              0.0              0.0             0.0              0.0
3              0.0              0.0             0.0              0.0
4              0.0              0.0             0.0              0.0

   telecommand_38  telecommand_39  telecommand_40  is_anomaly
0             0.0             0.0             0.0           0
1             0.0             0.0             0.0           0
2             0.0             0.0             0.0           0
3             0.0             0.0             0.0           0
4             0.0             0.0             0.0           0

[5 rows x 89 columns]

Valores nulos por columna:
id                 0
channel_1          0
channel_10         0
channel_11         0
channel_12         0
                  ..
telecommand_376    0
telecommand_38     0
telecommand_39     0
telecommand_40     0
is_anomaly         0
Length: 89, dtype: int64

Estadísticas descriptivas:
                 id     channel_1    channel_10    channel_11    channel_12  \
count  1.472832e+07  1.472832e+07  1.472832e+07  1.472832e+07  1.472832e+07
mean   7.364160e+06  1.380301e-01  6.403504e-09  4.395080e-10  2.557318e-01
std    4.251700e+06  4.428727e-03  1.492817e-06  2.467632e-07  5.158953e-02
min    0.000000e+00  9.217216e-02  0.000000e+00  0.000000e+00  1.225953e-01
25%    3.682080e+06  1.379103e-01  0.000000e+00  0.000000e+00  2.126752e-01
50%    7.364160e+06  1.379103e-01  0.000000e+00  0.000000e+00  2.547867e-01
75%    1.104624e+07  1.379103e-01  0.000000e+00  0.000000e+00  2.953383e-01
max    1.472832e+07  4.331867e-01  4.236102e-04  4.218630e-04  9.967715e-01

         channel_13    channel_14    channel_15    channel_16    channel_17  \
count  1.472832e+07  1.472832e+07  1.472832e+07  1.472832e+07  1.472832e+07
mean   3.094885e-01  2.720448e-01  1.210223e-01  7.610638e-01  2.927625e-01
```

```
std    5.081466e-02  9.578262e-02  6.525146e-02  7.655781e-02  5.072071e-02
min    1.721236e-01  7.845081e-02  1.286942e-03  7.355139e-03  1.422070e-01
25%    2.672639e-01  2.142259e-01  7.987051e-02  7.042502e-01  2.499678e-01
50%    3.078156e-01  2.594865e-01  1.185188e-01  7.612530e-01  2.906744e-01
75%    3.483671e-01  3.062542e-01  1.520136e-01  8.237714e-01  3.328890e-01
max    9.919286e-01  9.710565e-01  5.413337e-01  9.140906e-01  9.843958e-01

       …  telecommand_351  telecommand_352  telecommand_353  \
count  …     1.472832e+07     1.472832e+07     1.472832e+07
mean   …     1.629514e-06     1.765306e-06     1.901099e-06
std    …     1.276523e-03     1.328647e-03     1.378802e-03
min    …     0.000000e+00     0.000000e+00     0.000000e+00
25%    …     0.000000e+00     0.000000e+00     0.000000e+00
50%    …     0.000000e+00     0.000000e+00     0.000000e+00
75%    …     0.000000e+00     0.000000e+00     0.000000e+00
max    …     1.000000e+00     1.000000e+00     1.000000e+00

       telecommand_354  telecommand_36  telecommand_376  telecommand_38  \
count     1.472832e+07    1.472832e+07     1.472832e+07    1.472832e+07
mean      1.901099e-06    4.481163e-06     3.123234e-06    2.376374e-06
std       1.378802e-03    2.116871e-03     1.767265e-03    1.541547e-03
min       0.000000e+00    0.000000e+00     0.000000e+00    0.000000e+00
25%       0.000000e+00    0.000000e+00     0.000000e+00    0.000000e+00
50%       0.000000e+00    0.000000e+00     0.000000e+00    0.000000e+00
75%       0.000000e+00    0.000000e+00     0.000000e+00    0.000000e+00
max       1.000000e+00    1.000000e+00     1.000000e+00    1.000000e+00

       telecommand_39  telecommand_40    is_anomaly
count    1.472832e+07    1.472832e+07  1.472832e+07
mean     2.580063e-06    2.580063e-06  1.048391e-01
std      1.606256e-03    1.606256e-03  3.063460e-01
min      0.000000e+00    0.000000e+00  0.000000e+00
25%      0.000000e+00    0.000000e+00  0.000000e+00
50%      0.000000e+00    0.000000e+00  0.000000e+00
75%      0.000000e+00    0.000000e+00  0.000000e+00
max      1.000000e+00    1.000000e+00  1.000000e+00

[8 rows x 89 columns]
```

[ ]:
```
eda_summary(test, "Test Dataset")
```

```
--- Test Dataset ---
Shape: (521280, 88)

Columnas y tipos de datos:
id                    int64
```

```
channel_1          float32
channel_10         float32
channel_11         float32
channel_12         float32
                    …
telecommand_36     float64
telecommand_376      uint8
telecommand_38     float64
telecommand_39     float64
telecommand_40     float64
Length: 88, dtype: object

Primeras filas:
         id  channel_1  channel_10  channel_11  channel_12  channel_13  \
0  14728321    0.13791         0.0         0.0    0.218915    0.270384
1  14728322    0.13791         0.0         0.0    0.218915    0.270384
2  14728323    0.13791         0.0         0.0    0.218915    0.270384
3  14728324    0.13791         0.0         0.0    0.218915    0.270384
4  14728325    0.13791         0.0         0.0    0.218915    0.270384

   channel_14  channel_15  channel_16  channel_17  …  telecommand_350  \
0     0.63107    0.314563    0.786995    0.265045  …                0
1     0.63107    0.314563    0.786995    0.265045  …                0
2     0.63107    0.314563    0.786995    0.265045  …                0
3     0.63107    0.314563    0.786995    0.265045  …                0
4     0.63107    0.314563    0.786995    0.265045  …                0

   telecommand_351  telecommand_352  telecommand_353  telecommand_354  \
0                0                0                0              0.0
1                0                0                0              0.0
2                0                0                0              0.0
3                0                0                0              0.0
4                0                0                0              0.0

   telecommand_36  telecommand_376  telecommand_38  telecommand_39  \
0             0.0                0             0.0             0.0
1             0.0                0             0.0             0.0
2             0.0                0             0.0             0.0
3             0.0                0             0.0             0.0
4             0.0                0             0.0             0.0

   telecommand_40
0             0.0
1             0.0
2             0.0
3             0.0
4             0.0
```

```
[5 rows x 88 columns]

Valores nulos por columna:
id                 0
channel_1          0
channel_10         0
channel_11         0
channel_12         0
                  ..
telecommand_36     0
telecommand_376    0
telecommand_38     0
telecommand_39     0
telecommand_40     0
Length: 88, dtype: int64

Estadísticas descriptivas:
                 id      channel_1    channel_10    channel_11     channel_12  \
count  5.212800e+05  521280.000000  5.212800e+05  5.212800e+05  521280.000000
mean   1.498896e+07       0.137926  3.247337e-11  1.733109e-09       0.245810
std    1.504807e+05       0.001378  1.657859e-08  4.729760e-07       0.015413
min    1.472832e+07       0.092172  0.000000e+00  0.000000e+00       0.197078
25%    1.485864e+07       0.137910  0.000000e+00  0.000000e+00       0.232952
50%    1.498896e+07       0.137910  0.000000e+00  0.000000e+00       0.242309
75%    1.511928e+07       0.137910  0.000000e+00  0.000000e+00       0.257906
max    1.524960e+07       0.430109  8.463860e-06  1.321509e-04       0.281301

          channel_13     channel_14     channel_15     channel_16  \
count  521280.000000  521280.000000  521280.000000  521280.000000
mean        0.292378       0.640148       0.328910       0.815418
std         0.014934       0.016208       0.014169       0.025044
min         0.246989       0.575246       0.308786       0.779640
25%         0.279741       0.632757       0.316010       0.792512
50%         0.289100       0.637835       0.324676       0.810900
75%         0.304698       0.646288       0.340565       0.838481
max         0.324972       0.700417       0.425783       0.885460

          channel_17  …  telecommand_350  telecommand_351  telecommand_352  \
count  521280.000000  …         521280.0         521280.0         521280.0
mean        0.285674  …              0.0              0.0              0.0
std         0.016031  …              0.0              0.0              0.0
min         0.237907  …              0.0              0.0              0.0
25%         0.272584  …              0.0              0.0              0.0
50%         0.283137  …              0.0              0.0              0.0
75%         0.298214  …              0.0              0.0              0.0
max         0.323844  …              0.0              0.0              0.0

       telecommand_353  telecommand_354  telecommand_36  telecommand_376  \
```

```
count         521280.0    521280.000000   521280.000000         521280.0
mean               0.0         0.000002        0.000004              0.0
std                0.0         0.001385        0.001959              0.0
min                0.0         0.000000        0.000000              0.0
25%                0.0         0.000000        0.000000              0.0
50%                0.0         0.000000        0.000000              0.0
75%                0.0         0.000000        0.000000              0.0
max                0.0         1.000000        1.000000              0.0

       telecommand_38  telecommand_39  telecommand_40
count   521280.000000   521280.000000   521280.000000
mean         0.000002        0.000002        0.000002
std          0.001385        0.001385        0.001385
min          0.000000        0.000000        0.000000
25%          0.000000        0.000000        0.000000
50%          0.000000        0.000000        0.000000
75%          0.000000        0.000000        0.000000
max          1.000000        1.000000        1.000000

[8 rows x 88 columns]
```

[ ]:
```
train.columns # 41-46
```

```
Index(['id', 'channel_1', 'channel_10', 'channel_11', 'channel_12',
       'channel_13', 'channel_14', 'channel_15', 'channel_16', 'channel_17',
       'channel_18', 'channel_19', 'channel_2', 'channel_20', 'channel_21',
       'channel_22', 'channel_23', 'channel_24', 'channel_25', 'channel_26',
       'channel_27', 'channel_28', 'channel_29', 'channel_3', 'channel_30',
       'channel_31', 'channel_32', 'channel_33', 'channel_34', 'channel_35',
       'channel_36', 'channel_37', 'channel_38', 'channel_39', 'channel_4',
       'channel_40', 'channel_41', 'channel_42', 'channel_43', 'channel_44',
       'channel_45', 'channel_46', 'channel_47', 'channel_48', 'channel_49',
       'channel_5', 'channel_50', 'channel_51', 'channel_52', 'channel_53',
       'channel_54', 'channel_55', 'channel_56', 'channel_57', 'channel_58',
       'channel_59', 'channel_6', 'channel_60', 'channel_61', 'channel_62',
       'channel_63', 'channel_64', 'channel_65', 'channel_66', 'channel_67',
       'channel_68', 'channel_69', 'channel_7', 'channel_70', 'channel_71',
       'channel_72', 'channel_73', 'channel_74', 'channel_75', 'channel_76',
       'channel_8', 'channel_9', 'telecommand_244', 'telecommand_350',
       'telecommand_351', 'telecommand_352', 'telecommand_353',
       'telecommand_354', 'telecommand_36', 'telecommand_376',
       'telecommand_38', 'telecommand_39', 'telecommand_40', 'is_anomaly'],
      dtype='object')
```

[ ]:
```
span = 20_000
anomalies_range = []

for i in range(0, len(train), span):
    if 1 in train[i:i+span]["is_anomaly"].unique():
        print(f"Anomalía encontrada entre {i} y {i+span}")
        anomalies_range.append((i, i+span))
```

```
Anomalía encontrada entre 100000 y 120000
Anomalía encontrada entre 120000 y 140000
Anomalía encontrada entre 140000 y 160000
Anomalía encontrada entre 160000 y 180000
Anomalía encontrada entre 180000 y 200000
Anomalía encontrada entre 220000 y 240000
Anomalía encontrada entre 240000 y 260000
Anomalía encontrada entre 260000 y 280000
Anomalía encontrada entre 320000 y 340000
Anomalía encontrada entre 340000 y 360000
Anomalía encontrada entre 420000 y 440000
Anomalía encontrada entre 480000 y 500000
Anomalía encontrada entre 500000 y 520000
Anomalía encontrada entre 520000 y 540000
Anomalía encontrada entre 600000 y 620000
Anomalía encontrada entre 640000 y 660000
Anomalía encontrada entre 680000 y 700000
Anomalía encontrada entre 720000 y 740000
Anomalía encontrada entre 780000 y 800000
Anomalía encontrada entre 800000 y 820000
Anomalía encontrada entre 820000 y 840000
Anomalía encontrada entre 840000 y 860000
Anomalía encontrada entre 900000 y 920000
Anomalía encontrada entre 920000 y 940000
Anomalía encontrada entre 940000 y 960000
Anomalía encontrada entre 960000 y 980000
Anomalía encontrada entre 980000 y 1000000
Anomalía encontrada entre 1000000 y 1020000
Anomalía encontrada entre 1020000 y 1040000
Anomalía encontrada entre 1140000 y 1160000
Anomalía encontrada entre 1160000 y 1180000
Anomalía encontrada entre 1240000 y 1260000
Anomalía encontrada entre 1280000 y 1300000
Anomalía encontrada entre 1300000 y 1320000
Anomalía encontrada entre 1320000 y 1340000
Anomalía encontrada entre 1340000 y 1360000
Anomalía encontrada entre 1360000 y 1380000
Anomalía encontrada entre 1380000 y 1400000
Anomalía encontrada entre 1440000 y 1460000
```

```
Anomalía encontrada entre 1460000 y 1480000
Anomalía encontrada entre 1640000 y 1660000
Anomalía encontrada entre 1660000 y 1680000
Anomalía encontrada entre 1680000 y 1700000
Anomalía encontrada entre 1700000 y 1720000
Anomalía encontrada entre 1720000 y 1740000
Anomalía encontrada entre 1740000 y 1760000
Anomalía encontrada entre 1840000 y 1860000
Anomalía encontrada entre 1980000 y 2000000
Anomalía encontrada entre 2060000 y 2080000
Anomalía encontrada entre 2220000 y 2240000
Anomalía encontrada entre 2240000 y 2260000
Anomalía encontrada entre 2260000 y 2280000
Anomalía encontrada entre 2480000 y 2500000
Anomalía encontrada entre 2500000 y 2520000
Anomalía encontrada entre 2600000 y 2620000
Anomalía encontrada entre 2660000 y 2680000
Anomalía encontrada entre 2680000 y 2700000
Anomalía encontrada entre 2860000 y 2880000
Anomalía encontrada entre 2880000 y 2900000
Anomalía encontrada entre 3180000 y 3200000
Anomalía encontrada entre 3260000 y 3280000
Anomalía encontrada entre 3320000 y 3340000
Anomalía encontrada entre 3340000 y 3360000
Anomalía encontrada entre 3640000 y 3660000
Anomalía encontrada entre 3940000 y 3960000
Anomalía encontrada entre 3960000 y 3980000
Anomalía encontrada entre 4060000 y 4080000
Anomalía encontrada entre 4080000 y 4100000
Anomalía encontrada entre 4160000 y 4180000
Anomalía encontrada entre 4200000 y 4220000
Anomalía encontrada entre 4220000 y 4240000
Anomalía encontrada entre 4480000 y 4500000
Anomalía encontrada entre 4540000 y 4560000
Anomalía encontrada entre 4580000 y 4600000
Anomalía encontrada entre 4740000 y 4760000
Anomalía encontrada entre 4880000 y 4900000
Anomalía encontrada entre 4940000 y 4960000
Anomalía encontrada entre 4960000 y 4980000
Anomalía encontrada entre 5080000 y 5100000
Anomalía encontrada entre 5100000 y 5120000
Anomalía encontrada entre 5160000 y 5180000
Anomalía encontrada entre 5180000 y 5200000
Anomalía encontrada entre 5200000 y 5220000
Anomalía encontrada entre 5300000 y 5320000
Anomalía encontrada entre 5360000 y 5380000
Anomalía encontrada entre 5500000 y 5520000
Anomalía encontrada entre 5520000 y 5540000
```

```
Anomalía encontrada entre 5540000 y 5560000
Anomalía encontrada entre 5780000 y 5800000
Anomalía encontrada entre 5800000 y 5820000
Anomalía encontrada entre 5960000 y 5980000
Anomalía encontrada entre 5980000 y 6000000
Anomalía encontrada entre 6020000 y 6040000
Anomalía encontrada entre 6040000 y 6060000
Anomalía encontrada entre 6060000 y 6080000
Anomalía encontrada entre 6100000 y 6120000
Anomalía encontrada entre 6120000 y 6140000
Anomalía encontrada entre 6280000 y 6300000
Anomalía encontrada entre 6300000 y 6320000
Anomalía encontrada entre 6480000 y 6500000
Anomalía encontrada entre 6620000 y 6640000
Anomalía encontrada entre 6640000 y 6660000
Anomalía encontrada entre 6660000 y 6680000
Anomalía encontrada entre 6680000 y 6700000
Anomalía encontrada entre 7080000 y 7100000
Anomalía encontrada entre 7200000 y 7220000
Anomalía encontrada entre 7240000 y 7260000
Anomalía encontrada entre 7260000 y 7280000
Anomalía encontrada entre 7280000 y 7300000
Anomalía encontrada entre 7300000 y 7320000
Anomalía encontrada entre 7380000 y 7400000
Anomalía encontrada entre 7660000 y 7680000
Anomalía encontrada entre 7680000 y 7700000
Anomalía encontrada entre 7700000 y 7720000
Anomalía encontrada entre 7720000 y 7740000
Anomalía encontrada entre 7740000 y 7760000
Anomalía encontrada entre 7760000 y 7780000
Anomalía encontrada entre 7780000 y 7800000
Anomalía encontrada entre 7960000 y 7980000
Anomalía encontrada entre 8000000 y 8020000
Anomalía encontrada entre 8120000 y 8140000
Anomalía encontrada entre 8140000 y 8160000
Anomalía encontrada entre 8160000 y 8180000
Anomalía encontrada entre 8180000 y 8200000
Anomalía encontrada entre 8220000 y 8240000
Anomalía encontrada entre 8380000 y 8400000
Anomalía encontrada entre 8400000 y 8420000
Anomalía encontrada entre 8480000 y 8500000
Anomalía encontrada entre 8500000 y 8520000
Anomalía encontrada entre 8560000 y 8580000
Anomalía encontrada entre 8620000 y 8640000
Anomalía encontrada entre 8820000 y 8840000
Anomalía encontrada entre 8840000 y 8860000
Anomalía encontrada entre 8860000 y 8880000
Anomalía encontrada entre 8880000 y 8900000
```

```
Anomalía encontrada entre 9000000 y 9020000
Anomalía encontrada entre 9060000 y 9080000
Anomalía encontrada entre 9120000 y 9140000
Anomalía encontrada entre 9140000 y 9160000
Anomalía encontrada entre 9240000 y 9260000
Anomalía encontrada entre 9360000 y 9380000
Anomalía encontrada entre 9440000 y 9460000
Anomalía encontrada entre 9520000 y 9540000
Anomalía encontrada entre 9660000 y 9680000
Anomalía encontrada entre 9680000 y 9700000
Anomalía encontrada entre 9700000 y 9720000
Anomalía encontrada entre 9880000 y 9900000
Anomalía encontrada entre 9900000 y 9920000
Anomalía encontrada entre 9920000 y 9940000
Anomalía encontrada entre 9940000 y 9960000
Anomalía encontrada entre 9960000 y 9980000
Anomalía encontrada entre 9980000 y 10000000
Anomalía encontrada entre 10000000 y 10020000
Anomalía encontrada entre 10020000 y 10040000
Anomalía encontrada entre 10040000 y 10060000
Anomalía encontrada entre 10220000 y 10240000
Anomalía encontrada entre 10280000 y 10300000
Anomalía encontrada entre 10400000 y 10420000
Anomalía encontrada entre 10440000 y 10460000
Anomalía encontrada entre 10460000 y 10480000
Anomalía encontrada entre 10480000 y 10500000
Anomalía encontrada entre 10700000 y 10720000
Anomalía encontrada entre 10740000 y 10760000
Anomalía encontrada entre 10840000 y 10860000
Anomalía encontrada entre 10860000 y 10880000
Anomalía encontrada entre 10880000 y 10900000
Anomalía encontrada entre 10900000 y 10920000
Anomalía encontrada entre 10920000 y 10940000
Anomalía encontrada entre 10940000 y 10960000
Anomalía encontrada entre 11120000 y 11140000
Anomalía encontrada entre 11180000 y 11200000
Anomalía encontrada entre 11200000 y 11220000
Anomalía encontrada entre 11220000 y 11240000
Anomalía encontrada entre 11380000 y 11400000
Anomalía encontrada entre 11640000 y 11660000
Anomalía encontrada entre 11720000 y 11740000
Anomalía encontrada entre 11740000 y 11760000
Anomalía encontrada entre 11860000 y 11880000
Anomalía encontrada entre 11880000 y 11900000
Anomalía encontrada entre 11900000 y 11920000
Anomalía encontrada entre 11920000 y 11940000
Anomalía encontrada entre 12020000 y 12040000
Anomalía encontrada entre 12040000 y 12060000
```

```
Anomalía encontrada entre 12060000 y 12080000
Anomalía encontrada entre 12200000 y 12220000
Anomalía encontrada entre 12220000 y 12240000
Anomalía encontrada entre 12300000 y 12320000
Anomalía encontrada entre 12360000 y 12380000
Anomalía encontrada entre 12380000 y 12400000
Anomalía encontrada entre 12500000 y 12520000
Anomalía encontrada entre 12520000 y 12540000
Anomalía encontrada entre 12540000 y 12560000
Anomalía encontrada entre 12600000 y 12620000
Anomalía encontrada entre 12640000 y 12660000
Anomalía encontrada entre 12940000 y 12960000
Anomalía encontrada entre 13100000 y 13120000
Anomalía encontrada entre 13120000 y 13140000
Anomalía encontrada entre 13140000 y 13160000
Anomalía encontrada entre 13180000 y 13200000
Anomalía encontrada entre 13360000 y 13380000
Anomalía encontrada entre 13380000 y 13400000
Anomalía encontrada entre 13420000 y 13440000
Anomalía encontrada entre 13600000 y 13620000
Anomalía encontrada entre 13620000 y 13640000
Anomalía encontrada entre 13640000 y 13660000
Anomalía encontrada entre 13780000 y 13800000
Anomalía encontrada entre 13900000 y 13920000
Anomalía encontrada entre 14000000 y 14020000
Anomalía encontrada entre 14180000 y 14200000
Anomalía encontrada entre 14200000 y 14220000
Anomalía encontrada entre 14220000 y 14240000
Anomalía encontrada entre 14240000 y 14260000
Anomalía encontrada entre 14260000 y 14280000
Anomalía encontrada entre 14300000 y 14320000
Anomalía encontrada entre 14320000 y 14340000
Anomalía encontrada entre 14340000 y 14360000
Anomalía encontrada entre 14360000 y 14380000
Anomalía encontrada entre 14380000 y 14400000
Anomalía encontrada entre 14400000 y 14420000
Anomalía encontrada entre 14420000 y 14440000
Anomalía encontrada entre 14460000 y 14480000
Anomalía encontrada entre 14580000 y 14600000
Anomalía encontrada entre 14600000 y 14620000
Anomalía encontrada entre 14640000 y 14660000
```

[ ]:
```python
def plot_channels_interactive(df):
    channels = [41, 42, 43, 44, 45, 46]
    fig = go.Figure()
    for ch in channels:
        ch_name = f'channel_{ch}'
```

```
        fig.add_trace(go.Scatter(
            x=df['id'],
            y=df[ch_name],
            mode='lines',
            name=f'Canal {ch}',
            line=dict(width=2)
        ))
    fig.update_layout(
        title="Visualización interactiva de canales 41 al 46",
        xaxis_title="ID",
        yaxis_title="Valor",
        template="plotly_white"
    )
    fig.show()
```

[ ]:
```
plot_channels_interactive(train[anomalies_range[0][0]:anomalies_range[0][1]])
```

[ ]:
```
def plot_channels_interactive(df):
    channels = [f'channel_{ch}' for ch in range (41,47)]
    fig = go.Figure()

    anomalies = df[df['is_anomaly'] == 1]
    grouped_anomalies = []
    if not anomalies.empty:
        start = anomalies.iloc[0]['id']
        end = start
        for i in range(1, len(anomalies)):
            if anomalies.iloc[i]['id'] == anomalies.iloc[i - 1]['id'] + 1:
                end = anomalies.iloc[i]['id']
            else:
                grouped_anomalies.append((start, end))
                start = anomalies.iloc[i]['id']
                end = start
        grouped_anomalies.append((start, end))


    for ch in channels:
        fig.add_trace(go.Scatter(
            x=df['id'],
            y=df[ch],
            mode='lines',
            name=f'{ch}',
            line=dict(width=2)
        ))

    for start, end in grouped_anomalies:
        fig.add_shape(
```

```
            type="rect",
            x0=start - 0.5, x1=end + 0.5,
            y0=df[channels].min().min(),
            y1=df[channels].max().max(),
            line=dict(color="red", width=2),
            fillcolor="rgba(255, 0, 0, 0.2)",
            layer="below"
        )

    fig.update_layout(
        title="Visualización interactiva de canales 41 al 46 con anomalías␣
↪agrupadas",
        xaxis_title="ID",
        yaxis_title="Valor",
        template="plotly_white"
    )
    fig.show()
```

[ ]:
```
margin = 5_000

start_idx = max(0, anomalies_range[4][0] - margin)
end_idx = min(len(train), anomalies_range[8][1] + margin)

plot_channels_interactive(train.iloc[start_idx:end_idx])
```

5 SOTA

Antes de empezar a probar diferentes modelos y técnicas de preprocesado tratamos de alinearnos con el caso de estudio, ya no solo entendiendo los datos sino también intentando tomar decisiones informadas de cara a los modelos.

Además de revisar las diapositivas de la asignatura, refrescando lo visto en clase, también tratamos de dar un paso más allá tratando de comprender por nuestra cuenta qué hacen los autores de este nicho de investigación, completando estas ideas con las diapositivas y mejorar nuestra visión de las series temporales.

Sorprendentemente, vimos que había un amplio campo de investigación. Donde se llegaban a usar un gran conjunto de familias de deep learning para detección de anomalías en series temporales.

5.1 1. Redes generativas (GAN)

Modelos como MO-GAAL y AnoGAN aprenden la distribución de los datos normales para detectar desviaciones que puedan indicar una anomalía. Su principal ventaja es que suelen tener una alta precisión y una baja tasa de falsos positivos. Sin embargo, los autores comentan que el entrenamiento de estos modelos puede ser inestable y requiere bastante ajuste para evitar problemas como el colapso modal (como toda GAN).

También hay variantes como GANomaly y DCT-GAN, que mezclan ideas de autoencoders y GANs.

Estas variantes incorporan transformaciones multiescala que permiten capturar mejor la estructura de las secuencias temporales.

5.2 2. Autoencoders avanzados

MemAAE es un autoencoder que incluye un módulo de memoria para evitar que el modelo reconstruya bien las anomalías. Al combinar tareas de reconstrucción y predicción, consigue resultados muy buenos, llegando a F1 cercanos a 0.90 (según los autores y en su entorno).

Otro enfoque interesante es DAGMM, que utiliza un autoencoder junto con un modelo de mezcla gaussiana en el espacio latente. Además, se han probado combinaciones como VAE con transformers, o autoencoders con convoluciones dilatadas (TCN-AE), que amplían el campo receptivo y mejoran la sensibilidad ante anomalías sutiles.

5.3 3. Modelos con transformers

Anomaly Transformer introduce un tipo especial de atención que analiza cómo se relaciona cada punto de la serie con el resto. La idea es que las anomalías suelen tener relaciones más locales, mientras que los datos normales se asocian con toda la serie.

TranAD combina transformers con entrenamiento adversarial y mecanismos de autoajuste. Esto lo hace más robusto, especialmente con series largas y anomalías muy sutiles.

Dual-TF va un paso más allá usando dos transformers en paralelo: uno en el dominio temporal y otro en el espectral (por ejemplo, usando transformada de Fourier). Esta combinación permite detectar patrones anómalos más complejos.

TS2Vec aplica aprendizaje contrastivo multiescala para generar representaciones útiles sin necesidad de etiquetas. Estas representaciones luego se pueden usar para tareas de detección de anomalías con buenos resultados.

5.4 4. Otros enfoques

Una alternativa a LSTM son las redes convolucionales temporales (TCN), que funcionan muy bien con datos de alta frecuencia y permiten capturar tanto patrones locales como dependencias a largo plazo.

Por último, otra opción sería combinar varios modelos (por ejemplo, un autoencoder y un transformer). Esta combinación permitiría aprovechar las fortalezas de cada uno y mejorar la detección general.

Año	Modelo	Familia	Enlace al paper
2018	MO-GAAL	GAN	Paper
2017	AnoGAN	GAN	Paper
2018	GANomaly	GAN + AE	Paper
2021	DCT-GAN	GAN + Multiscale	Paper

Año	Modelo	Familia	Enlace al paper
2020	MemAE / MemAAE	AE + Memory	Paper
2018	DAGMM	AE + GMM	Paper
2022	Anomaly Transformer	Transformer	Paper
2022	TranAD	Transformer + GAN	Paper
2024	Dual-TF	Dual Transformer (Time/Freq)	Paper
2021	TS2Vec	Contrastive / Transformer	Paper
2020	TCN	Temporal CNN	Paper

Cabe descatar que no todo esto esta enfocado a las series temporales ni a los datos con los que vamos a trabajar, pero si que consideramos que son tecnologías útiles para nuestro proyecto o al menos papers que nos ayudaron a bajar el nivel de abstracción y mejorar nuestra conocimiento en la materia.

6 Baseline

Teniendo en mente el SOTA, empezamos a probar nuestras propias aproximaciones.

6.1 IsolationForest

En esta sección se construye un modelo baseline para la detección de anomalías utilizando el algoritmo **Isolation Forest**, una técnica bastante común para detectar outliers en conjuntos de datos de alta dimensionalidad.

Pasos realizados:

1. **Selección de características**:
   Se seleccionan únicamente las columnas que comienzan con `channel_`, que representan las señales o lecturas de sensores del dataset de entrenamiento.
2. **Entrenamiento del modelo**:
   Se entrena un modelo de `IsolationForest` con un 1% de contaminación (`contamination=0.01`), lo que implica que se espera que aproximadamente el 1% de los datos sean anómalos. Se utiliza `n_jobs=-1` para aprovechar todos los núcleos disponibles y acelerar el proceso.
3. **Detección y visualización de anomalías**:
   Las observaciones clasificadas como anómalas (`-1`) se almacenan y se visualizan para un subconjunto de los datos (`channel_41`), lo que permite una primera evaluación visual de la calidad del modelo.
4. **Aplicación al conjunto de test**:
   El modelo entrenado se aplica al conjunto de test, marcando cada instancia como normal (`0`) o anómala (`1`). Los resultados se guardan en un archivo CSV para subirlo al kaggle.

[ ]:
```
features = [col for col in train.columns if col.startswith('channel_')]
X = train[features]

model = IsolationForest(contamination=0.01, random_state=42, n_jobs=-1)
```

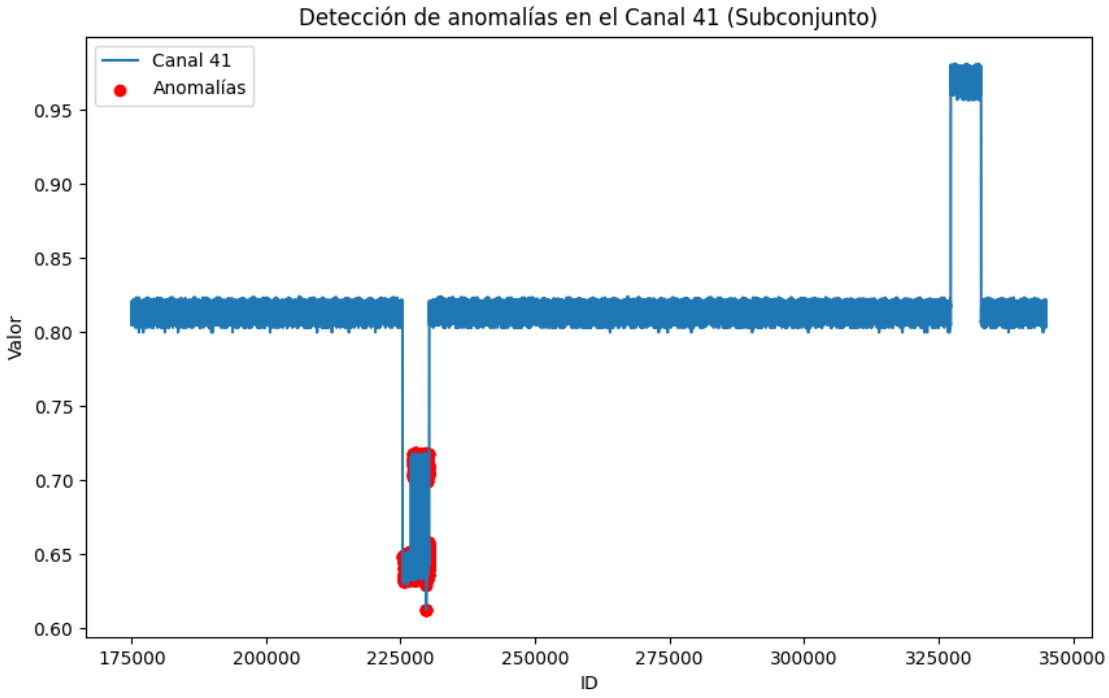
```
train['anomaly_score'] = model.fit_predict(X)

anomalies = train[train['anomaly_score'] == -1]
print(f"Total de anomalías detectadas: {len(anomalies)}")
```

```
Total de anomalías detectadas: 147284
```

[ ]:
```
subset = train.iloc[start_idx:end_idx]
anomalies_subset = anomalies[(anomalies['id'] >= subset['id'].min()) &␣
 ↪(anomalies['id'] <= subset['id'].max())]

plt.figure(figsize=(10, 6))
plt.plot(subset['id'], subset['channel_41'], label='Canal 41')
plt.scatter(anomalies_subset['id'], anomalies_subset['channel_41'],␣
 ↪color='red', label='Anomalías')
plt.title('Detección de anomalías en el Canal 41 (Subconjunto)')
plt.xlabel('ID')
plt.ylabel('Valor')
plt.legend()
plt.show()
```



[ ]:
```
test = pd.read_parquet('../data/test.parquet')

features = [col for col in test.columns if col.startswith('channel_')]
```

```
X_test = test[features]
test['is_anomaly'] = model.predict(X_test)

test['is_anomaly'] = test['is_anomaly'].apply(lambda x: 1 if x == -1 else 0)

test[['id', 'is_anomaly']].to_csv('../out/predictions_IsolationForest_baseline.
 ↪csv', index=False)
```

predictions_IsolationForest_baseline.csv

Complete · Corti · 1mo ago

Score: 0.004

Los resultados fueron peores de lo esperado.

7 Advanced experiments

7.1 Data

[ ]:
```
train_df = pd.read_parquet("../data/train.parquet")
test_df = pd.read_parquet("../data/test.parquet")
```

Consideramos que uno de los principales problemas que tuvimos en el baseline fue que tratamos con todas las features de `channels`, así que para estos nuevos experimentos nos centraremos en un pequeño conjunto de canales, del 41-46 siguiendo la recomendación del kaggle.

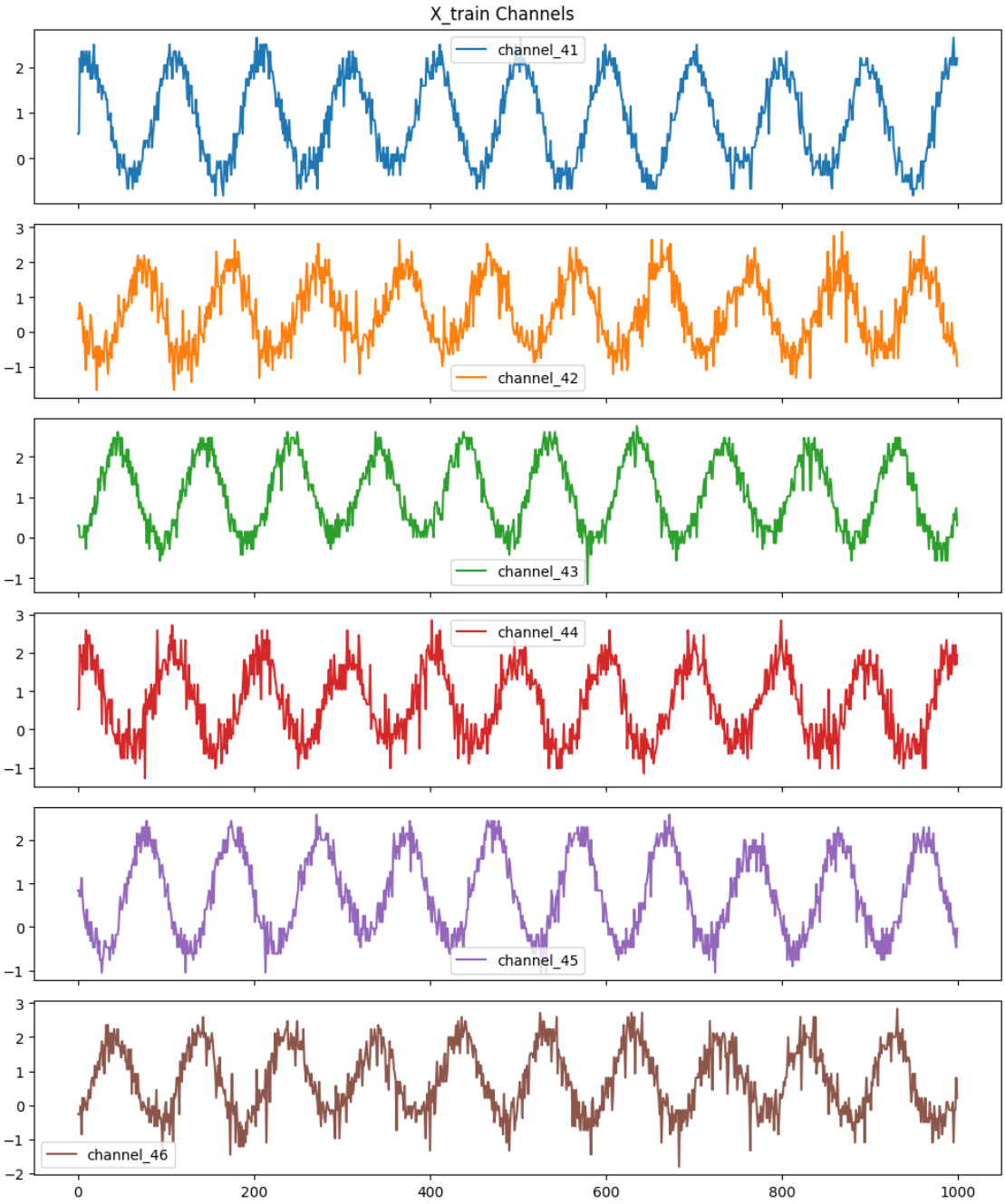
Por otro lado, nos quedamos con los datos que no tienen anomalías, esto se debe a que así será más sencillo para los modelos interpretar las secuencias normales y anómalas, generando errores más altos al reconstruir fragmentos anomalos ya que nunca los han visto previamente.

[ ]:
```
CHANNELS = [f"channel_{i}" for i in range(41, 47)]

scaler = StandardScaler()
X_train = scaler.fit_transform(train_df.loc[train_df["is_anomaly"] == 0,␣
 ↪CHANNELS])
X_test = scaler.transform(test_df[CHANNELS])
```

[ ]:
```
X_train_df = pd.DataFrame(X_train[:1_000], columns=CHANNELS)

X_train_df.plot(subplots=True, figsize=(10, 12), title="X_train Channels")
plt.tight_layout()
plt.show()
```



Por otro lado, tras pelearnos en la sección anterior y teniendo como referencia las prácticas de la asignatura, decidimos estrablecer ventanas para procesar los datos. Estás ventanas aumentaron bastante la demanda de RAM de cara a los futuros entrenamientos, llevando al límite nuestros recursos.

Nos vimos obligados a trabajar con ventanas de 5, ya que trabajar con ventanas más grandes hacía que se muriera el kernel del notebook o directamente no terminaba nunca de ejecutarse congelando nuestros ordenadores.

[ ]:
```
timesteps = 5

def create_sequences(data, timesteps):
    return np.array([data[i:i+timesteps] for i in range(len(data)-timesteps)])

X_train_seq = create_sequences(X_train, timesteps)
X_test_seq = create_sequences(X_test, timesteps)
```

7.2 Data 2

En esta sección se lleva a cabo un preprocesamiento avanzado de los datos, orientado a capturar mejor las características temporales.

Tras probar con el conjunto de datos anterior pensamos que podríamos dar un paso más mejorando el preprocesado inicial, ya que anteriormente habíamos hecho simplemente una estandarización. Esta propueta consistía en aplicar transformaciones wavelet a cada canal ([41, 46]) con la ligera esperanza de eliminar ruido y resaltar patrones relevantes.

[ ]:
```
train_df = pd.read_parquet("../data/train.parquet")
test_df = pd.read_parquet("../data/test.parquet")

CHANNELS = [f"channel_{i}" for i in range(41, 47)]

scaler = StandardScaler()
X_train_raw = scaler.fit_transform(train_df.loc[train_df["is_anomaly"] == 0,␣
 ↪CHANNELS])
X_test_raw = scaler.transform(test_df[CHANNELS])

timesteps = 5

def create_sequences(data, timesteps):
    return np.array([data[i:i+timesteps] for i in range(len(data)-timesteps)])

X_train_seq = create_sequences(X_train_raw, timesteps)
X_test_seq = create_sequences(X_test_raw, timesteps)


def apply_wavelet_seq(data_seq, wavelet='db4', level=1):
    n_samples, n_timesteps, n_channels = data_seq.shape
    transformed_seq = np.zeros_like(data_seq)

    for sample in tqdm(range(n_samples), desc="Wavelet transform"):
        for ch in range(n_channels):
            ts = data_seq[sample, :, ch]
            coeffs = pywt.wavedec(ts, wavelet=wavelet, level=level)
            ts_recon = pywt.waverec(coeffs, wavelet=wavelet)
            ts_recon = ts_recon[:n_timesteps]
            transformed_seq[sample, :, ch] = ts_recon
```

```
    return transformed_seq


X_train_seq = apply_wavelet_seq(X_train_seq)
X_test_seq = apply_wavelet_seq(X_test_seq)
```

```
Wavelet transform: 100%|      | 13184212/13184212 [30:23<00:00, 7230.39it/s]
Wavelet transform: 100%|      | 521275/521275 [01:11<00:00, 7270.72it/s]
```

7.3 Models

Motivados por la filosofía de reconstrucción del estado del arte, empezamos a implementar nuestros propios modelos.

Cabe destacar que estos experimentos que hemos dejado en el notebook son con el dataset de `Data` no `Data 2`, usar wavelet no supuso mejores resultados. A lo largo de la sección, se harán algunas menciones a este dataset hablando de los resultados obtenidos.

7.3.1 AE

[ ]:
```
input_dim = X_train.shape[1]
inputs = Input(shape=(input_dim,))
encoded = Dense(16, activation='relu')(inputs)
encoded = Dropout(0.2)(encoded)
bottleneck = Dense(8, activation='relu')(encoded)
encoded = Dropout(0.2)(encoded)
decoded = Dense(16, activation='relu')(bottleneck)
decoded = Dropout(0.2)(decoded)
outputs = Dense(input_dim, activation='linear')(decoded)
autoencoder = Model(inputs, outputs)

autoencoder.compile(optimizer='adam', loss='mse')
autoencoder.summary()
```

```
I0000 00:00:1746259659.497903   19849 gpu_device.cc:2019] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 3831 MB memory:  -> device: 0,
name: NVIDIA GeForce RTX 2060, pci bus id: 0000:2d:00.0, compute capability: 7.5
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 6)	0
dense (Dense)	(None, 16)	112
dropout (Dropout)	(None, 16)	0

```
 dense_1 (Dense)                 (None, 8)                      136

 dense_2 (Dense)                 (None, 16)                     144

 dropout_2 (Dropout)             (None, 16)                       0

 dense_3 (Dense)                 (None, 6)                      102


Total params: 494 (1.93 KB)

Trainable params: 494 (1.93 KB)

Non-trainable params: 0 (0.00 B)
```

[ ]:
```
autoencoder.fit(X_train, X_train, epochs=5, batch_size=1024, shuffle=True,␣
↪verbose=1)
```

```
Epoch 1/5
12876/12876             33s 3ms/step
- loss: 0.2135
Epoch 2/5
12876/12876             32s 2ms/step
- loss: 0.1551
Epoch 3/5
12876/12876             28s 2ms/step
- loss: 0.1481
Epoch 4/5
12876/12876             31s 2ms/step
- loss: 0.1457
Epoch 5/5
12876/12876             31s 2ms/step
- loss: 0.1450
<keras.src.callbacks.history.History at 0x7f3ee93fedd0>
```

[ ]:
```
X_test_reconstructed = autoencoder.predict(X_test, batch_size=1024)
per_channel_error = np.abs(X_test - X_test_reconstructed)
z_scores = (per_channel_error - per_channel_error.mean(axis=0)) /␣
↪(per_channel_error.std(axis=0) + 1e-8)
aggregated_scores = z_scores.max(axis=1)
```

```
510/510             1s 2ms/step
```

[ ]:
```python
def prune_anomalies(anoms, min_len=10):
    anoms = anoms.copy()
    labels, num = label(anoms)
    for i in range(1, num+1):
        idx = np.where(labels == i)[0]
        if len(idx) < min_len:
            anoms[idx] = 0
    return anoms
```

[ ]:
```python
threshold = np.percentile(aggregated_scores, 98)
binary_scores = (aggregated_scores > threshold).astype(int)

is_anomaly = prune_anomalies(binary_scores, min_len=10)
```

Parece que quedarnos con los 6 canales es una idea acertada. Aquí empezamos a ver los primeros frutos de nuesta intuición reconstruyendo la información de la secuencia, el siguiente paso fue tratar de desarrollar un modelo más complejo capaz de retener mejor la información y mejorar el score.

7.3.2 LSTM Autoencoder

Una de las primeras aproximaciones que barajamos fue usar RNN (motivados por la literatura), para que a través de la memoria pudieramos retener mejor la información de los patrones en los canales, esto lo logramos añadiendo capas de LSTM.

[ ]:
```python
l2_reg = l2(0.001)

inputs = Input(shape=(timesteps, input_dim))

encoded = LSTM(64,
               return_sequences=False,
               dropout=0.2,
               recurrent_dropout=0.2,
               kernel_regularizer=l2_reg)(inputs)

decoded = RepeatVector(timesteps)(encoded)

decoded = LSTM(64,
               return_sequences=True,
               dropout=0.2,
               recurrent_dropout=0.2,
               kernel_regularizer=l2_reg)(decoded)

outputs = TimeDistributed(Dense(input_dim, kernel_regularizer=l2_reg))(decoded)
```

```
model = Model(inputs, outputs)
model.compile(optimizer='adam', loss='mse')
model.summary()
```

```
I0000 00:00:1746262047.637015    5114 gpu_device.cc:2019] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 4047 MB memory:  -> device: 0,
name: NVIDIA GeForce RTX 2060, pci bus id: 0000:2d:00.0, compute capability: 7.5
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 5, 6)	0
lstm (LSTM)	(None, 64)	18,176
repeat_vector (RepeatVector)	(None, 5, 64)	0
lstm_1 (LSTM)	(None, 5, 64)	33,024
time_distributed (TimeDistributed)	(None, 5, 6)	390

Total params: 51,590 (201.52 KB)

Trainable params: 51,590 (201.52 KB)

Non-trainable params: 0 (0.00 B)

[ ]:
```
model.fit(X_train_seq, X_train_seq, epochs=2, batch_size=1024, shuffle=True,␣
↪verbose=1)
```

```
2025-05-03 10:47:29.139364: W
external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
1582105440 exceeds 10% of free system memory.
2025-05-03 10:47:30.432748: W
external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
1582105440 exceeds 10% of free system memory.
2025-05-03 10:47:31.517668: W
external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
1582105440 exceeds 10% of free system memory.
2025-05-03 10:47:31.914349: W
```

```
external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
1582105440 exceeds 10% of free system memory.

Epoch 1/2
12876/12876             1282s
99ms/step - loss: 0.1693
Epoch 2/2
12876/12876             1310s
102ms/step - loss: 0.1083

<keras.src.callbacks.history.History at 0x7f73bfb9e620>
```

```
[ ]: X_test_seq_reconstructed = model.predict(X_test_seq, batch_size=1024)
per_channel_error_seq = np.abs(X_test_seq - X_test_seq_reconstructed)
z_scores_seq = (per_channel_error_seq - per_channel_error_seq.mean(axis=0)) /␣
  ↪(per_channel_error_seq.std(axis=0) + 1e-8)
aggregated_scores = z_scores_seq.max(axis=2).mean(axis=1)
```

```
510/510             7s 15ms/step
```

```
[ ]: threshold = np.percentile(aggregated_scores, 98)
binary_scores = (aggregated_scores > threshold).astype(int)

is_anomaly = prune_anomalies(binary_scores, min_len=10)
```

Aquí vimos el primer salto más importante del desarrollo, donde superamos a la mayoría de participantes de kaggle y nos dió un puesto entre los 8 mejores equipos en la competición.

7.3.3 BLSTM Autoencoder

Para mejorar aún más el modelo decidimos usar BLSTM, de está manera la retención de la información sería más eficiente.

```
[ ]: l2_reg = l2(0.001)

inputs = Input(shape=(timesteps, input_dim))

encoded = Bidirectional(LSTM(64,
                             return_sequences=False,
                             dropout=0.2,
                             recurrent_dropout=0.2,
                             kernel_regularizer=l2_reg))(inputs)

decoded = RepeatVector(timesteps)(encoded)
```

```
decoded = Bidirectional(LSTM(64,
                             return_sequences=True,
                             dropout=0.2,
                             recurrent_dropout=0.2,
                             kernel_regularizer=l2_reg))(decoded)

outputs = TimeDistributed(Dense(input_dim, kernel_regularizer=l2_reg))(decoded)

model = Model(inputs, outputs)
model.compile(optimizer='adam', loss='mse')
model.summary()
```

```
I0000 00:00:1746340386.743767    4011 gpu_device.cc:2019] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 4047 MB memory:  -> device: 0,
name: NVIDIA GeForce RTX 2060, pci bus id: 0000:2d:00.0, compute capability: 7.5
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 5, 6)	0
bidirectional (Bidirectional)	(None, 128)	36,352
repeat_vector (RepeatVector)	(None, 5, 128)	0
bidirectional_1 (Bidirectional)	(None, 5, 128)	98,816
time_distributed (TimeDistributed)	(None, 5, 6)	774

Total params: 135,942 (531.02 KB)

Trainable params: 135,942 (531.02 KB)

Non-trainable params: 0 (0.00 B)

[ ]:
```
model.fit(X_train_seq, X_train_seq, epochs=2, batch_size=1024, shuffle=True,␣
↪verbose=1)
```

```
2025-05-04 08:33:08.507060: W
external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
1582105440 exceeds 10% of free system memory.
```

```
2025-05-04 08:33:09.691133: W
external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
1582105440 exceeds 10% of free system memory.
2025-05-04 08:33:10.945088: W
external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
1582105440 exceeds 10% of free system memory.
2025-05-04 08:33:11.280637: W
external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
1582105440 exceeds 10% of free system memory.

Epoch 1/2
12876/12876            2658s
206ms/step - loss: 0.1277
Epoch 2/2
12876/12876            2591s
201ms/step - loss: 0.0630

<keras.src.callbacks.history.History at 0x7fa1663293c0>
```

```
[ ]: X_test_seq_reconstructed = model.predict(X_test_seq, batch_size=1024)
per_channel_error_seq = np.abs(X_test_seq - X_test_seq_reconstructed)
z_scores_seq = (per_channel_error_seq - per_channel_error_seq.mean(axis=0)) /␣
  ↪(per_channel_error_seq.std(axis=0) + 1e-8)
aggregated_scores = z_scores_seq.max(axis=2).mean(axis=1)
```

```
510/510            20s 38ms/step
```

```
[ ]: threshold = np.percentile(aggregated_scores, 98)
binary_scores = (aggregated_scores > threshold).astype(int)

is_anomaly = prune_anomalies(binary_scores, min_len=10)
```

Este fue otro salto bastante importante, gracías a esta mejora nos colocamos en el top 5 de mejores equipos.

7.3.4 Conv1D-LSTM + Attention

Tomando como referencia el estado del arte de la sección anterior decidimos probar con convoluciones. Esto fue motivado ya que leimos que al usar convoluciones y atención podríamos mejorar las relaciones entre canales.

```
[ ]: input_dim = X_train_seq.shape[2]
latent_dim = 64

inputs = Input(shape=(timesteps, input_dim))
```

```
x = Conv1D(filters=32, kernel_size=3, padding='same', activation='relu')(inputs)
x = LSTM(latent_dim, return_sequences=True)(x)
x = LayerNormalization()(x)

attention = Attention()([x, x])
context = Concatenate()([x, attention])
encoded = LSTM(latent_dim, return_sequences=False)(context)

decoded = RepeatVector(timesteps)(encoded)
decoded = LSTM(latent_dim, return_sequences=True)(decoded)
decoded = TimeDistributed(Dense(input_dim))(decoded)

model = Model(inputs, decoded)
model.compile(optimizer=Adam(1e-3), loss='mse')
model.summary()
```

```
I0000 00:00:1747549798.894817     925 gpu_device.cc:2019] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 4047 MB memory:  -> device: 0,
name: NVIDIA GeForce RTX 2060, pci bus id: 0000:2d:00.0, compute capability: 7.5
```

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 5, 17)	0	-
conv1d (Conv1D)	(None, 5, 32)	1,664	input_layer[0][0]
lstm (LSTM)	(None, 5, 64)	24,832	conv1d[0][0]
layer_normalization (LayerNormalizatio…	(None, 5, 64)	128	lstm[0][0]
attention (Attention)	(None, 5, 64)	0	layer_normalizat… layer_normalizat…
concatenate (Concatenate)	(None, 5, 128)	0	layer_normalizat… attention[0][0]
lstm_1 (LSTM)	(None, 64)	49,408	concatenate[0][0]
repeat_vector (RepeatVector)	(None, 5, 64)	0	lstm_1[0][0]
lstm_2 (LSTM)	(None, 5, 64)	33,024	repeat_vector[0]…

```
time_distributed      (None, 5, 17)            1,105   lstm_2[0][0]
(TimeDistributed)
```

```
Total params: 110,161 (430.32 KB)
```

```
Trainable params: 110,161 (430.32 KB)
```

```
Non-trainable params: 0 (0.00 B)
```

[ ]:
```
history = model.fit(
    X_train_seq, X_train_seq,
    epochs=1,
    batch_size=1024,
    verbose=1
)
```

```
2025-05-10 11:41:19.961650: W
external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
1265684160 exceeds 10% of free system memory.
```

```
10301/10301             193s
19ms/step - loss: 5.2631e-04 - val_loss: 2.8885e-04
```

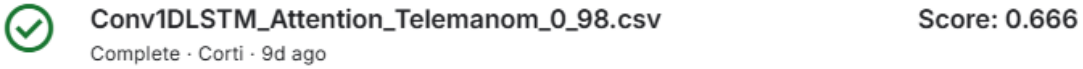
[ ]:
```
X_test_seq_reconstructed = model.predict(X_test_seq, batch_size=1024)
per_channel_error_seq = np.abs(X_test_seq - X_test_seq_reconstructed)
z_scores_seq = (per_channel_error_seq - per_channel_error_seq.mean(axis=0)) /␣
↪(per_channel_error_seq.std(axis=0) + 1e-8)
aggregated_scores = z_scores_seq.max(axis=2).mean(axis=1)
```

```
510/510             3s 5ms/step
```

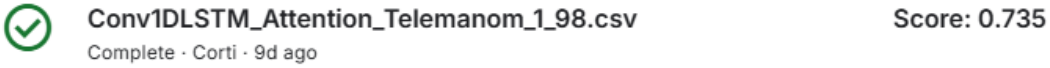
[ ]:
```
threshold = np.percentile(aggregated_scores, 98)
binary_scores = (aggregated_scores > threshold).astype(int)

is_anomaly = prune_anomalies(binary_scores, min_len=10)
```

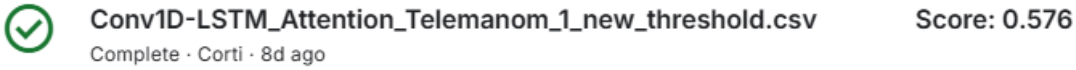
Las perdidas fueron bastante buenas, las más bajas de todos los modelos probados, esto nos hizo comenter overfitting. A pesar de que en la celda de arriba hayamos dejado la ejecución con 1 epoch, incialmente hicimos 2 epochs como en los modelos anteriores donde obtuvimos:



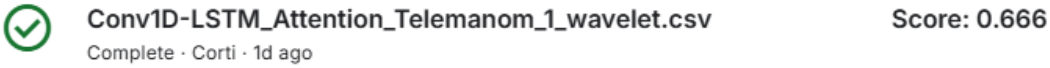
Este resultado es inferior al modelo anterior. Sin embargo, volviendo a ejecutar el fit pero con 1 epoch obtuvimos:



Curiosamente es el mismo score que en el BLSTM-AE. Ya que este modelo nos ha ofrecido muy buenos resultados, probamos a modificar el umbral. Este úmbral viene definido por el método prune_anomalies donde siempre habíamos utilizado el percentil 98.



Ya que no obtuvimos resultados favorables, volvimos al percentil 98. Ya que este modelo nos dió buenos resultados mantuvimos nuestra mejor configuración y lo entrenamos con los datos preprocesados con wavelet:



Obtuvimos peores resultados, volvimos a los otros datos y tratamos de buscar otras alternativas.

7.3.5 Conv1D-BLSTM + Attention

Motivados por los resultados anteriores, pensamos que añadir capas bidireccionales daría un mejor resultado, asi que lo probamos.

[ ]:
```
input_dim = X_train.shape[1]
latent_dim = 64

inputs = Input(shape=(timesteps, input_dim))
x = Conv1D(filters=32, kernel_size=3, padding='same', activation='relu')(inputs)
x = Bidirectional(LSTM(latent_dim, return_sequences=True))(x)
x = LayerNormalization()(x)

attention = Attention()([x, x])
context = Concatenate()([x, attention])
encoded = Bidirectional(LSTM(latent_dim, return_sequences=False))(context)

decoded = RepeatVector(timesteps)(encoded)
decoded = Bidirectional(LSTM(latent_dim, return_sequences=True))(decoded)
decoded = TimeDistributed(Dense(input_dim))(decoded)

model = Model(inputs, decoded)
model.compile(optimizer=Adam(1e-3), loss='mse')
model.summary()
```

```
I0000 00:00:1746891678.408197    4520 gpu_device.cc:2019] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 4047 MB memory:  -> device: 0,
name: NVIDIA GeForce RTX 2060, pci bus id: 0000:2d:00.0, compute capability: 7.5
```

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 7, 6)	0	-
conv1d (Conv1D)	(None, 7, 32)	608	input_layer[0][0]
bidirectional (Bidirectional)	(None, 7, 128)	49,664	conv1d[0][0]
layer_normalization (LayerNormalizatio…	(None, 7, 128)	256	bidirectional[0]…
attention (Attention)	(None, 7, 128)	0	layer_normalizat… layer_normalizat…
concatenate (Concatenate)	(None, 7, 256)	0	layer_normalizat… attention[0][0]
bidirectional_1 (Bidirectional)	(None, 128)	164,352	concatenate[0][0]
repeat_vector (RepeatVector)	(None, 7, 128)	0	bidirectional_1[…
bidirectional_2 (Bidirectional)	(None, 7, 128)	98,816	repeat_vector[0]…
time_distributed (TimeDistributed)	(None, 7, 6)	774	bidirectional_2[…

Total params: 314,470 (1.20 MB)

Trainable params: 314,470 (1.20 MB)

Non-trainable params: 0 (0.00 B)

[ ]:
```
history = model.fit(
    X_train_seq, X_train_seq,
    epochs=1,
    batch_size=1024,
    verbose=1
)
```

```
2025-05-10 17:41:20.161672: W
external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
1771957488 exceeds 10% of free system memory.
2025-05-10 17:41:21.859623: W
external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
1771957488 exceeds 10% of free system memory.
2025-05-10 17:41:23.791693: W
external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
1771957488 exceeds 10% of free system memory.
2025-05-10 17:41:24.101066: W
external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
1771957488 exceeds 10% of free system memory.
```

```
Epoch 1/2
```

```
I0000 00:00:1746891692.135849    4684 cuda_dnn.cc:529] Loaded cuDNN version
90300
```

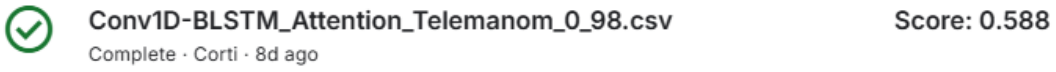
```
10301/10301             0s 30ms/step
- loss: 0.1391
```

[ ]:
```
X_test_seq_reconstructed = model.predict(X_test_seq, batch_size=1024)
per_channel_error_seq = np.abs(X_test_seq - X_test_seq_reconstructed)
z_scores_seq = (per_channel_error_seq - per_channel_error_seq.mean(axis=0)) /␣
↪(per_channel_error_seq.std(axis=0) + 1e-8)
aggregated_scores = z_scores_seq.max(axis=2).mean(axis=1)
```

```
510/510             5s 9ms/step
```

[ ]:
```
threshold = np.percentile(aggregated_scores, 98)
binary_scores = (aggregated_scores > threshold).astype(int)

is_anomaly = prune_anomalies(binary_scores, min_len=10)
```



Los resultados fueron peores, esto sugiere que la arquitectura podría no ser la más adecuada para este caso de estudio. Pensamos que añadir BLSTM añade una complejidad extra al modelo que es inecesaria y que no aporta nada a la detección de patrones.

Además este modelo junto a Conv1D-LSTM y Conv2D-LSTM rozaba el límite de nuestros recursos, si vemos los logs del entrenamiento podemos ver: "Allocation of 1771957488 exceeds 10% of free system memory.". Entrenar se volvió un suplicio, esto nos hizo ir mucho más lento y nos empezaba a limitar nuestra creatividad.

7.3.6 Conv2D-LSTM + Attention

Para tratar de dar un paso más, probamos a usar `Conv2D`. Partimos de la suposición de que al estar procesando señales y existir una estrecha relación con las imágenes podríamos conseguir buenos resultados.

[ ]:
```
input_dim = X_train.shape[1]

inputs = Input(shape=(timesteps, input_dim))

x = Reshape((timesteps, input_dim, 1))(inputs)

x = Conv2D(filters=16, kernel_size=(3, 3), padding="same", activation="relu")(x)
x = Conv2D(filters=32, kernel_size=(3, 3), padding="same", activation="relu")(x)

x = Reshape((timesteps, -1))(x)
x = LSTM(64, return_sequences=True)(x)
x = LayerNormalization()(x)

attn = Attention()([x, x])
x = Concatenate()([x, attn])
encoded = LSTM(64, return_sequences=False)(x)

decoded = RepeatVector(timesteps)(encoded)
decoded = LSTM(64, return_sequences=True)(decoded)
decoded = TimeDistributed(Dense(input_dim))(decoded)

model = Model(inputs, decoded)
model.compile(optimizer='adam', loss='mse')
model.summary()
```

```
Model: "functional_1"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer_2 (InputLayer)	(None, 5, 6)	0	-
reshape_4 (Reshape)	(None, 5, 6, 1)	0	input_layer_2[0]…
conv2d_4 (Conv2D)	(None, 5, 6, 16)	160	reshape_4[0][0]

conv2d_5 (Conv2D)	(None, 5, 6, 32)	4,640	conv2d_4[0][0]
reshape_5 (Reshape)	(None, 5, 192)	0	conv2d_5[0][0]
lstm_3 (LSTM)	(None, 5, 64)	65,792	reshape_5[0][0]
layer_normalizatio… (LayerNormalizatio…	(None, 5, 64)	128	lstm_3[0][0]
attention_1 (Attention)	(None, 5, 64)	0	layer_normalizat… layer_normalizat…
concatenate_1 (Concatenate)	(None, 5, 128)	0	layer_normalizat… attention_1[0][0]
lstm_4 (LSTM)	(None, 64)	49,408	concatenate_1[0]…
repeat_vector_1 (RepeatVector)	(None, 5, 64)	0	lstm_4[0][0]
lstm_5 (LSTM)	(None, 5, 64)	33,024	repeat_vector_1[…
time_distributed_1 (TimeDistributed)	(None, 5, 6)	390	lstm_5[0][0]

Total params: 153,542 (599.77 KB)

Trainable params: 153,542 (599.77 KB)

Non-trainable params: 0 (0.00 B)

[ ]:
```
history = model.fit(
    X_train_seq, X_train_seq,
    epochs=1,
    batch_size=1024,
    validation_data=(X_val_seq, X_val_seq),
    verbose=1
)
```

```
2025-05-17 08:28:22.183997: W
external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
1265684160 exceeds 10% of free system memory.
2025-05-17 08:28:23.317819: W
external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
```

```
1265684160 exceeds 10% of free system memory.
2025-05-17 08:28:24.273930: W
external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
1265684160 exceeds 10% of free system memory.
2025-05-17 08:28:24.529508: W
external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
1265684160 exceeds 10% of free system memory.
I0000 00:00:1747463309.020101    2689 cuda_dnn.cc:529] Loaded cuDNN version
90501
```

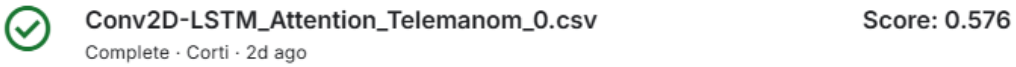
```
10301/10301             205s
19ms/step - loss: 0.1071 - val_loss: 0.0010
```

[ ]:
```
X_test_seq_reconstructed = model.predict(X_test_seq, batch_size=1024)
per_channel_error_seq = np.abs(X_test_seq - X_test_seq_reconstructed)
z_scores_seq = (per_channel_error_seq - per_channel_error_seq.mean(axis=0)) /␣
 ↪(per_channel_error_seq.std(axis=0) + 1e-8)
aggregated_scores = z_scores_seq.max(axis=2).mean(axis=1)
```

```
510/510             3s 5ms/step
```

[ ]:
```
threshold = np.percentile(aggregated_scores, 98)
binary_scores = (aggregated_scores > threshold).astype(int)

is_anomaly = prune_anomalies(binary_scores, min_len=10)
```



A pesar de que teníamos mucha ilusión con este experimento, no obtuvimos buenos resultados.

8 Results

Para testear tuvimos muchos problemas, inicialmente probamos a guardarnos un conjunto de test haciendo un `split` de train para sacar métricas. Sin embargo, llevamos tan al límite nuestros recursos que en muchas ocasiones al procesar los resultados se nos moría el kernel, teníamos lo justo para entrenar y generar los CSV.

Podríamos haber guardado los pesos y ejecutar métricas a parte pero como los resultados del kaggle superaban a la mayoría de equipos pensamos que lo mejor sería tratar de usar todos los recursos existentes en subir el score público de la competición asumiendo que vamos por buen camino.

El código de los CSVs es:

[ ]:
```
aligned_ids = test_df["id"].iloc[timesteps:].reset_index(drop=True)

submission_df = pd.DataFrame({
    "id": aligned_ids,
```

```
    "is_anomaly": is_anomaly
})

submission_df.to_csv("../out/X.csv", index=False)
```

Simplemente cargamos el conjunto de `test` y las anomalías generadas al final de cada entrenamiento.

8.1 Summary

Model	Public Score
Conv1D-LSTM_Attention_Telemanom_1_98.csv	0.735
BLSTM_AE_Telemanom_0.98.csv	0.735
Conv1D-LSTM_Attention_Telemanom_0_98.csv	0.666
Conv1D-LSTM_Attention_Telemanom_1_wavelet.csv	0.666
LSTM_AE_Telemanom_0.98.csv	0.657
Conv1D-BLSTM_Attention_Telemanom_0.csv	0.588
Conv1D-LSTM_Attention_Telemanom_1_new_threshold.csv	0.576
Conv2D-LSTM_Attention_Telemanom_0.csv	0.576
AE_Telemanom_1.csv	0.277
IsolationForest.csv	0.004

9 Conclusions

Estamos satisfechos con nuestros resultados, a día en el que se esta redactando este documento somos los 5º de 53 equipos. Teniendo en cuenta que seguramente tendremos menos experiencia que los demás participantes creemos que hemos hecho un gran trabajo.

Nos habría gustado probar más ideas que teníamos en mente, entre ellas que pasaría si aumentamos el tamaño de ventana. Actualmente, estos experimentos han sido con un tamaño de 5 motivados por la falta de recursos, creemos que si hubieramos podido desarrollar en equipos más potentes podríamos haber mejorado nuestros resultados probando con ventanas más grandes.

Otro problema fueron las features, estamos usando solo los canales $41 - 46$ cuando hay varias columnas como `telecommand_` que si hubieramos podido explorar habría reducido las posibilidades de overfitting y nos habría permitido ejecutar más de 2 epochs.

Algo que también habría sido muy interesante es crear una especie de ensemble combinando los autoencoders para reconstruir y otro modelo para predecir, todo en el mismo pipeline (ya sobrepasabamos en un 10% los resursos, no llevamos ni a implementarlo). En cuanto al preprocesado, pensabamos que aplicar transformadas de wavelet podría ser útil descomponiendo la señal en diferentes niveles de resolución temporal-frecuencia pero no terminó de funcionar. Seguramente existan mejores aproximaciones para tratar los datos, pero de manera similar a los modelos al ser tantos datos el tiempo de ejecución de disparaba y el kernel moría.

También habría estado bien poder hacer búsqueda de hiperparámetros, tratando de llevar al limite nuestros modelos. Teníamos una carpeta dedicada a este tuning con un contenedor docker preparado para hacer búsquedas con Ray Tune desde JupyterLab, pero finalmente no llegamos a hacer la búsqueda ya que tampoco sabíamos como plantear las métricas a minimizar debido a que

los modelos se encargan de reconstruir. Podríamos haber minimizado en base al loss que devolvía pera si ya de normal había overfitting debido al tamaño tan reducido de ventana y a que solo estabamos usando 6 canales no consideramos que hubiese aportado mucho sin aumentar las features o el tamaño de ventana (no podíamos por falta de recursos).

Con más tiempo, sería interesante reproducir alguno de los papers del estado del arte aprovechando que la mayoría tienen Github. De ese modo podríamos ganar fácilmente la competición.

Si tuvieramos que escoger alguno de los modelos anteriores, nos quedamos con el `Conv1D-LSTM_Attention_Telemanom_1_98.csv`, consideramos que es más robusto que el `BLSTM_AE_Telemanom_0.98.csv` apesar de tener el mismo score, gracias a las convoluciones y a las capas de attention, permitiendole interpretar mejor los patrones intrínsecos de los datos.