PyDTS: Python Deep Time-series Simulation a toolkit for industrial time-series prediction

1. Introduction

Python Deep Timeseries Simulation (PyDTS) is a toolkit for capturing unknown and possibly non-linear relations between input and output values. The current version includes statistical, machine-learning, and deep-learning approaches for time-series modelling. The aim is to provide a better understanding for time-series modelling problems such as: forecasting, denoising, non-linear SISO and MIMO modelling, degradation modelling, and anomaly detection. It is clear, that this toolkit cannot be anywhere near the capabilities of commercial software but will hopefully provide a better understanding due to the freely available source code. The toolkit is obviously not complete; thus, suggestions are always welcome.

1.1. Publication and Citation

The PyDTS toolkit is part of the following survey paper and tries to replicate the presented architectures and approaches.

1.2. Dependencies

The requirements of the PyDTS toolkit are summarized in the requirements.txt data file. In detail, the PyDTS Toolkit was implemented using the following dependencies:

- Python 3.8
- Tensorflow 2.5
- Scikit-Lean 1.0
- Numpy
- Pandas
- Scipy

1.3. Datasets

The toolkit utilizes the following publicly available datasets, which have been pre-processed and reshaped for the user's convenience and can be found under /data. The datasets have been clustered into one test dataset for showing the working principle of the toolkit and six tutorials covering different applications for time-series modelling. It should be noted that all datasets are measured, and no synthetic data has been used.

1) Tutorial 1 (Denoising) based on AMPDs2 dataset [1]: https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/FIE0S4
2) Tutorial 2 (Forecasting) based on dataset [2]: https://www.kaggle.com/datasets/fedesoriano/electric-power-consumption
3) Tutorial 3 (Non-linear modeling) based on Electric Motor Temperature [3]: https://www.kaggle.com/datasets/wkirgsn/electric-motor-temperature
4) Tutorial 4 (Anomaly detection) based on Ford dataset [4]: https://www.timeseriesclassification.com/description.php?Dataset=FordA
5) Tutorial 5 (Degradation modeling) based on Battery aging [5]: https://kilthub.cmu.edu/articles/dataset/eVTOL_Battery_Dataset/14226830/2
6) Tutorial 6 (Cross Domain modeling) based on vehicle data [6]: https://ieee-dataport.org/open-access/battery-and-heating-data-real-driving-cycles

If the user wants to utilize own datasets, data can be provided in '.csv', '.xlsx', and '.mat' formats at the moment. Data templates can be found under \data.

1.4. Folder Structure

The folder structure of the PyDTS system can be found below. Users should mainly work in \data for configuration of new datasets and in \setup for saving parameter configurations as well as in \results for obtaining saved calculation results. The folder \src contains all source code and should only be modified if the user wishes to implement new functionalities.

Table 1: PyDTS folder structure.

PyDTS	**Folder**	**Subfolder**	**Content**
|--	data		Contains the datasets
|--	docu		Contains the documentation
|--	mdl		Contains all trained models
|--	results		Contains all results
|--	setup		Preconfigured setup sheets
|--	src		
	|--	data	Contains functions for loading data
	|--	external	Contains external functions
	|--	general	Contains general and helper functions
	|--	model	Contains functions for modeling

1.5. Limitations

Since the toolkit is still under development there are several things that need to be improved, are not yet implemented, or lack verification. In the following a list of know issues and limitations is provided:

- The transfer functions are not yet reviewed and have shown inconsistent behaviour.
- The data balancing function is currently not working and must be revised.
- There are minor issues for displaying and formatting units.

2. Architecture

The aim of the PyDTS toolkit is to model the input and output relation of an unknown system, namely $\hat{y} = f(X)$ where $X \in \mathbb{R}^{T \times M}$ is a feature input vector of T time-steps and M features and y is a time-series output. A more generalised architecture is illustrated below:

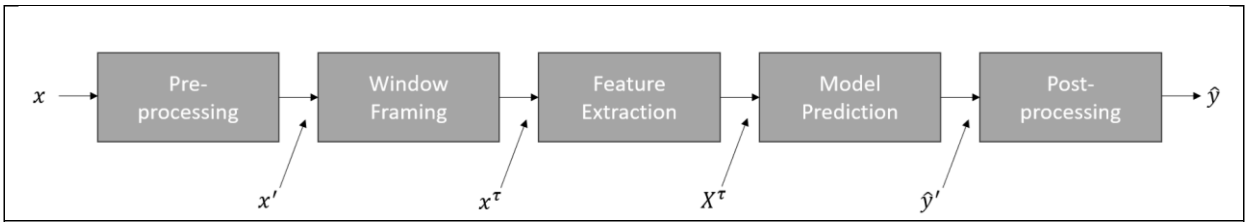


Figure 1: Proposed architecture as implemented in the PyDTS *toolkit.*

As can be seen the general architecture consists of five steps namely pre-processing (e.g. resampling or filtering), window framing, feature extraction, model prediction, and post-processing (e.g. limiting the output). The architecture is used for all applications which are described below.

2.1. Applications

The toolkit aims to cover different applications as they usually appear in industrial settings for time-series modelling. The following applications have been covered:

1) Denoising (Regression): $y(t) = \sum_M x_m(t) + \epsilon(t)$
2) Forecasting (Regression): $y(t) = \phi y(t-1) + \beta x(t) + \epsilon(t)$
3) Non-linear modelling (Regression): $y(t) = f\big(x(t)\big)$
4) Anomaly detection (Classification): $y(t) = \Theta\Big(f\big(x(t)\big)\Big)$
5) Degradation modelling (Regression): $y(t) = y_0 + \beta x(t) + \epsilon(t)$

2.2. Operating Modes

The toolkit offers three different operating modes. The operating modes can be controlled using 'setupExp['sim']' option in the start.py file. The following three option are available. First, classical training and testing where a model is trained using a training and validation dataset and tested on a testing dataset respectively. Second, hyperparameters optimization where free model parameters are optimally tuned using scikit GridSearchCV for machine learning models and keras hyperband tuner for deep learning models utilizing tensorboard. Third, general parameters can be optimised using a 2D grid search available in 'optiGrid.py'.

2.3. Implemented Models

The toolkit as three different types of models implemented namely models based on Signal Processing (SP), e.g. State-Space (SS) models or Transfer Function (TF) models, models based on Machine Learning (ML), e.g. Support Vector Machine (SVM) or Random Forest (RF), models based on Deep Learning (DL), e.g. Long Short Term Memory (LSTM) or Convolutional Neural Network (CNN).

3. Parameters

The PyDTS toolkit offers a set of pre-implemented option to configure the architecture and the models. The complete list, including description and possible values, are tabulated in Table 2.

Table 2: Complete options for the configuration of the PyDTS tool.

Experimental Settings				
Name	**Values**	**Default**	**Notes**	**Units**
Name	String	Test	Name of the simulation setup	-
Author	String	Pascal Schirmer	Author name of the toolkit	-
Sim	0, 1, 2	0	0) simulation, 1) optimisation hyperparameters, 2) optimising grid	-
Gpu	0, 1	1	0) cpu, 1) gpu	-
Warn	0, 1, 2, 3	3	0) all msg are logged, 1) INFO not logged, 2) INFO and WARN not logged, 3) disabled	-
Method	0, 1, 2, 3	2	0) 1-fold with data split, 1) k-fold with cross validation, 2) transfer learning with different datasets, 3) id based	-
trainBatch	0, 1, 2	0	0) all no batching, 1) fixed batch size (see data batch parameter), 2) id based	-
Kfold	Integer	10	number of folds for method 1)	-
Train	0, 1	0	0) no training (trying to load model), 1) training new model (or retraining)	-
Test	0, 1	1	0) no testing, 1) testing	-
Save	0, 1	0	0) results are not saved, 1) results are saved	-
Log	0, 1	0	0) no data logging, 1) logging input data	-
Plot	0, 1	1	0) plotting inactive, 1) plotting active	-
Data Settings				
Type	1, 2, 3	1	data input type: 1) 'xlsx', 2) 'csv', 3) 'mat'	-
Batch	Integer	100000	number of samples fed at once to training	-
Shuffle	True, False	False	False: no shuffling, True: shuffling data when splitting	-
rT	[0, 1]	0.9	training proportion (0, 1)	-
rV	[0, 1]	0.2	validation proportion (0, 1) as percentage from training proportion	-
idT	Integer	2	list of testing ids for method 3)	-
idV	Integer	2	list of validation ids for method 3)	-
Train	String	dataTest_1	name of training datasets (multiple)	-
Test	String	dataTest_3	name of testing datasets (one)	-
Val	String	dataTest_4	name of validation dataset (one)	-
Inp	String	[]	names of the input variables (X) if empty all	-
Out	String	[‘Solar’, ‘Wind’]	names of the output variables (y)	-
fs	Float	1/900	sampling frequency	Hz
Lim	Integer	0	0) data is not limited, x) limited to x samples	-
weightNorm	0, 1	0	0) separate normalisation per input/output channel, 1) weighted normalisation	-
inpNorm	0, 1, 2, 3	3	normalising input values (X): 0) None, 1) -1/+1, 2) 0/1, 3) avg/sig	-
outNorm	0, 1, 2, 3	2	normalising output values (y): 0) None, 1) -1/+1, 2) 0/1, 3) avg/sig	-
inpNoise	Float	0	adding gaussian noise to input	dB
outNoise	Float	0	adding gaussian noise to output	dB
inpFil	0, 1	0	filtering input data (X): 0) None, 1) Median	-
outFil	0, 1	0	filtering output data (y): 0) None, 1) Median	-
inpFilLen	Integer	61	filter length input data (samples)	-
outFilLen	Integer	61	filter length output data (samples)	-

Threshold	Float	500	0) no threshold x) threshold to transform regressio into classification data	-
Balance	Integer	0	0) no balancing 1) balancing based classes, x) balancing based on x bins	-
General Settings				
Method	0, 1	0	0) regression, 1) classification	-
Solver	SP, ML, DL	DL	solver 1) 'SP': Signal Processing, 2) 'ML': Machine Learning, 3) 'DL': Deep Learning	-
Model	String	DNN	possible models 1) SP: State Space (SS), Transfer Function (TF), 2) ML: RF, KNN, SVM, 3) DL: CNN, LSTM, DNN	-
Lag	Integer	0	lagging between input (X) and output (y) in samples	-
Frame	0, 1	1	0) no framing, 1) framing	-
Feat	0, 1, 2, 3	0	0) raw data values, 1) statistical features (frame based), 2) statistical features (input based), 3) input and frame based features	-
Init	0, 1	0	0) no initial values 1) adding initial values from y	-
Window	Integer	96	window length (samples)	-
Overlap	Integer	95	overlap between consecutive windows (no overlap during test if -1)	-
Outseq	Integer	0	0) seq2point, x) length of the subsequence in samples	-
yFocus	Integer	95	focus point for seq2point (average if -1)	-
nDim	2, 3	3	input dimension for model 2) or 3)	-
outMin	Float	-1e9	limited output values (minimum)	-
outMax	Float	+1e9	limited output values (maximum)	-
Model Settings				
Batch	Integer	1000	batch size for training and testing	-
Epoch	Integer	100	number of epochs for training	-
Patience	Integer	15	number of epochs as patience during training	-
Valsteps	Integer	50	number of validation steps	-
Shuffle	True, False	False	shuffling data before training (after splitting data)	-
Verbose	0, 1, 2	2	level of detail for showing training information (0 silent)	-
Loss	String	Mae	loss function 1) mae, 2) mse, 3) BinaryCrossentropy 4) KLDivergence	-
Metric	String	Mse	loss metric 1) mae, 2) mse, 3) BinaryCrossentropy 4) KLDivergence	-
Opt	String	Adam	solver 1) Adam, 2) RMSprop, 3) SGD	-
Lr	Float	1e-4	learning rate	-
Beta1	[0, 1]	0.9	first moment decay	-
Beta2	[0, 1]	0.999	second moment decay	-
Eps	Float	1e-8	small constant for stability	-
Rho	Float	0.9	discounting factor for the history/coming gradient	-
mom	Float	0.0	momentum	-

4. Quick Start

In the following chapter a set of reference results is provided using the "default.txt" setup files provide in \setup. For a first test run use start.py to calculate the results presented below, which are already provided as pretrained models under \mdl. Therefore, training is disabled in start.py. The problem considered in the test example aims to model the regenerative power (MW) of solar and wind generate in Germany. As input feature standard weather data are available, namely solar irradiance (W/m^2), wind speed (m/s), and temperature (℃). The average results for different models in terms of Mean Absolute Error (MAE) are provided in Table 3. All results have been obtained using the default parameters of the toolkit.

Table 3: Average results for the example dataset.

Output	**Machine Learning**			**Deep Learning**		
	SVM	**KNN**	**RF**	**DNN**	**LSTM**	**CNN**
Solar	2717.31	3092.78	725.93	996.56	**685.31**	800.04
Wind	2937.16	3603.52	2360.53	2096.60	**1988.47**	2006.85
Average	2827.24	3348.15	1543.23	1546.58	**1336.89**	1403.45

As can be seen from the table above, DL approaches consistently outperform ML approaches for all architecture. Furthermore, it should be noted that the LSTM DL model shows the best results due to its ability to capture temporal information. The simulation output contains three different plots. The first one provides an analysis on the input features using violin plots, a heatmap of the input and output features, as well as a feature ranking calculated by RF. The second one plots the ground-truth against the predicted values and evaluates mean error and error distribution. The third one shows the predicted results as well as the error with respect to the ground-truth for both raw values and labels.

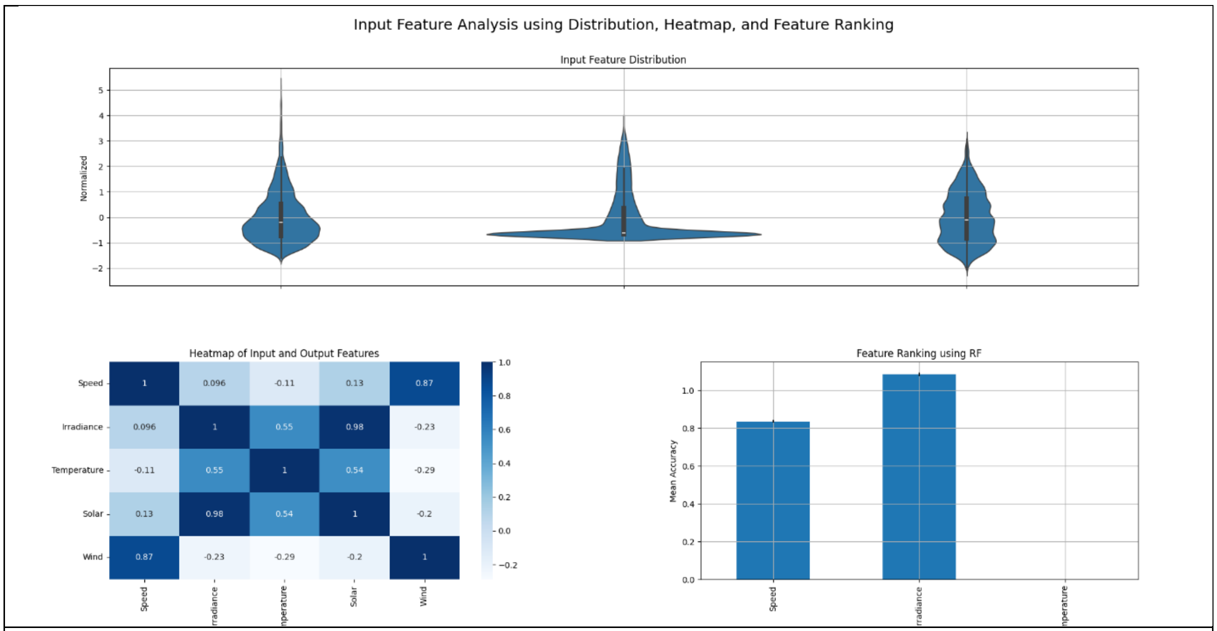


Figure 1: Input feature analysis, heatmap, and feature ranking using RF classifier.

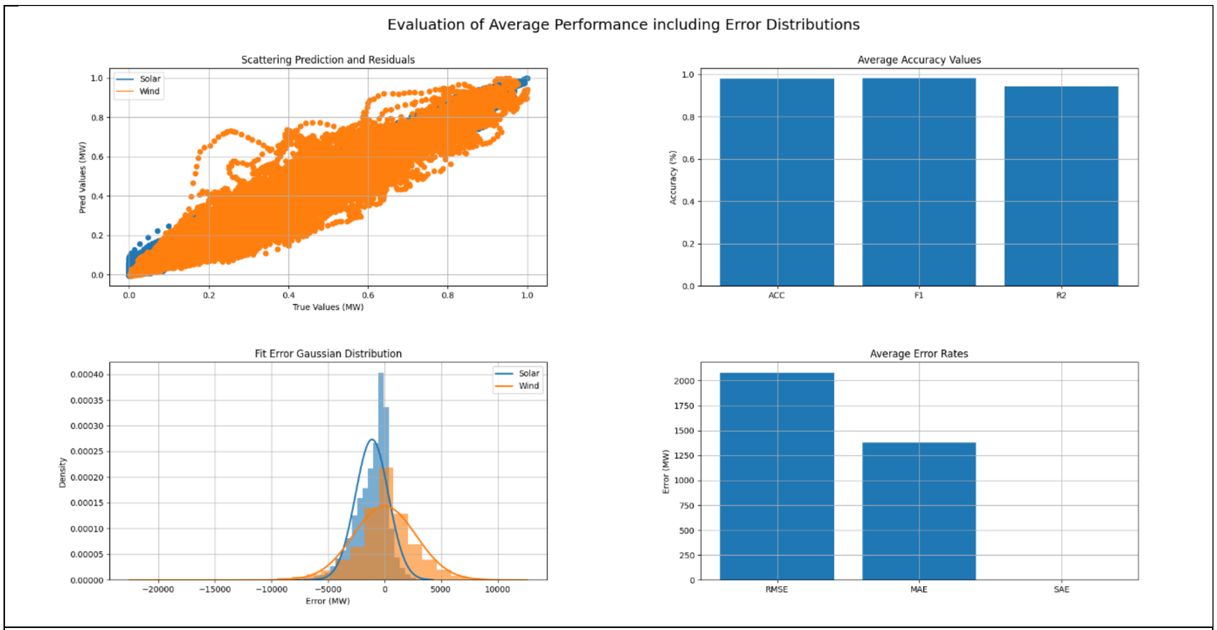


Figure 2: Scattering results, Gaussian error distribution, and average accuracy values.

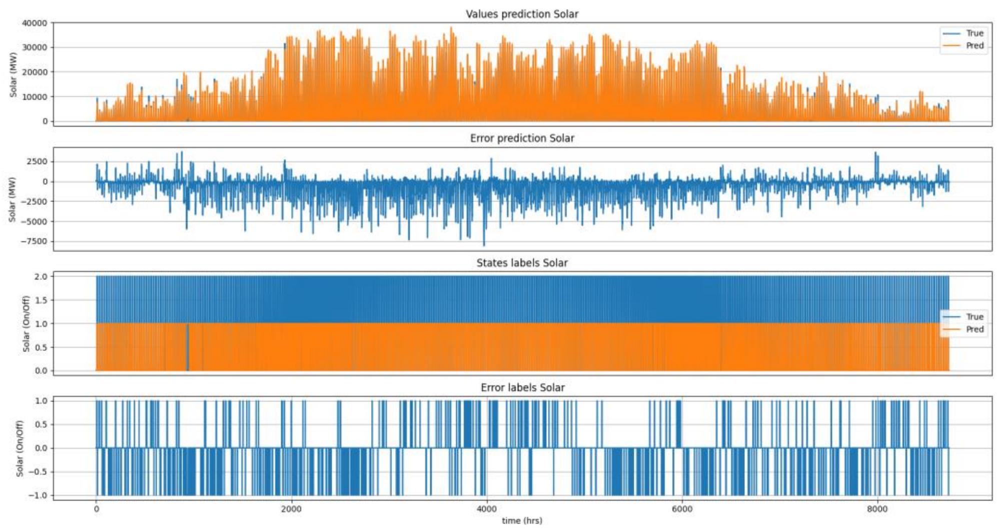


Figure 3: Ground-truth and predicted values for solar irradiance.

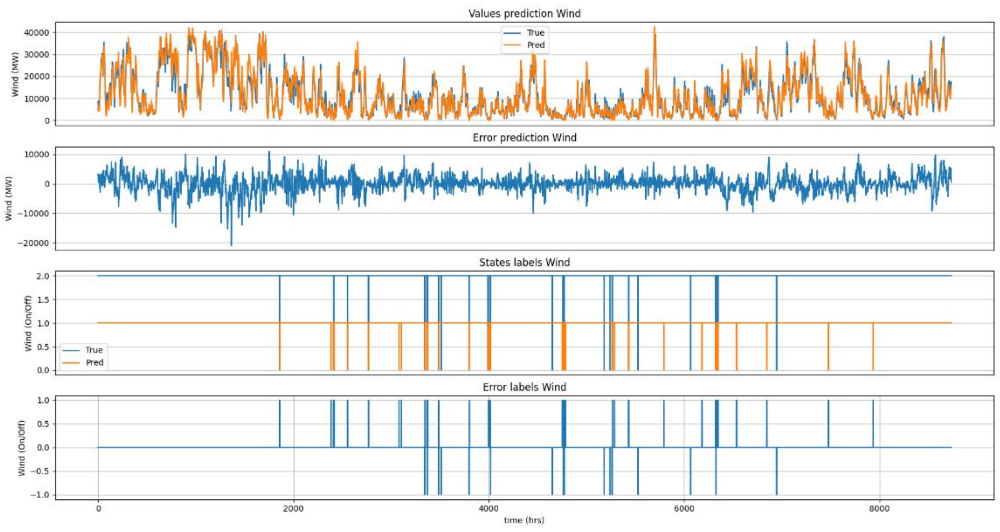


Figure 4: Ground-truth and predicted values for wind speed.

5. Results

In the following chapter a set of reference results is provided for each of the Tutorials. Due to file size not all of the trained models are not provided but can be easily retrained using the respective setup files under \setup. For each tutorial short background information on the problem is provided. Furthermore, it should be noted that the results are limited to the three deep learning model, namely DNN, LSTM, and CNN.

5.1. Denoising / Disaggregation (Tutorial 1)

Energy disaggregation is a task where appliance energy signatures are extracted from the aggregated data. Since, multiple signals are extracted from a single observation, it is a single channel blind source separation problem, i.e. a problem with very high signal to noise ratio. The problem can be mathematically formulated as follows:

$$y(t) = f(x_m(t), \epsilon(t)) = \sum_{m=1}^{M} x_m(t) + \epsilon(t) \quad (1)$$

$$\hat{x}_m(t) = f^{-1}(y(t)) \quad (2)$$

where $y(t)$ is the aggregated signal, $x_m(t)$ is the appliance signal, and $\epsilon(t)$ is additional noise from unknown devices. The goal is denoise the signal $y(t)$ in such a way to extract a desired appliance signature $\hat{x}_m(t)$ using an inverse function $f^{-1}(\cdot)$, which will be learned by a deep learning algorithm. In this tutorial we use the AMPDs2 dataset, which contains two years of data from a Canadian household. The input features are the aggregated active, reactive, and apparent power as well as the aggregated current. The output features are the per appliance current draw. All parameters for training and testing the algorithm can be found in the correspondent setup files under \setup. The results for different performance metrics and deep learning algorithms are tabulated in Table 4.

Table 4: Average results for the Tutorial 1 using deep learning approaches.

	TECA	RMSE	MAE	MAX
DNN	95.89%	0.75	0.10	42.05
LSTM	96.63%	0.66	0.08	37.40
CNN	96.45%	0.69	0.09	35.81

As can be seen LSTM outperforms all other deep-learning models except for the maximum error of the CNN. One exemplary output for the disaggregated signals is provided below.

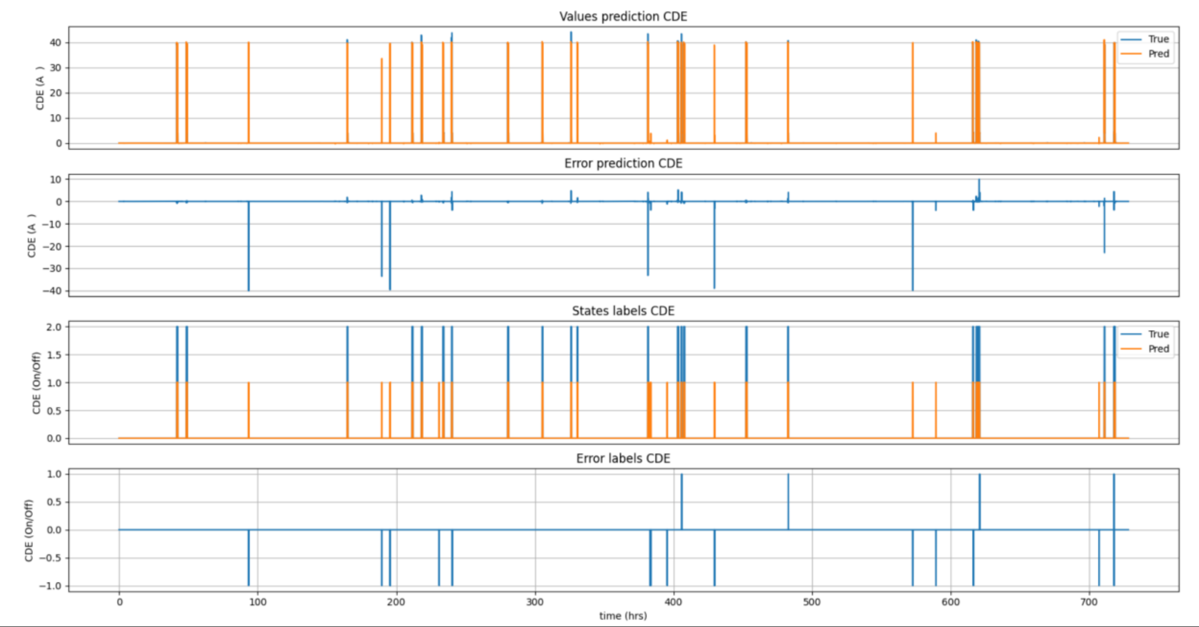


Figure 5: Example output for the energy disaggregation task.

5.2. Forecasting (Tutorial 2)

Load forecasting is a task where future values, e.g. energy consumption or power draw, are predicted based on previous values of the same time-series. The aim is to model temporal information based on previous samples and accurately predict future values. The problem can be mathematically formulated as follows:

$$y(t) = \phi y(t-1) + \beta x(t) + \epsilon(t) \quad (3)$$

$$\hat{y}(t) = f\,(y(t{:}\,t-W)) \quad (4)$$

where $y(t)$ is the power consumption signal, $x\,(t)$ are signals with additional information, e.g. weather, and $\epsilon(t)$ is stochastic noise. In this tutorial we will focus on only using the W previous samples from $y(t{:}\,t-W)$ to predict the current sample $\hat{y}\,(t)$ using a function $f\,(\cdot)$, which will be learned by a deep learning algorithm. We will use the Electric Power Consumption dataset [2], which provides three different power readings. All three power readings serve as input feature, while the first one is being used as time lagged output feature, e.g. the output feature is the time shifted variant of the first input feature. The goal is to predict 4 hrs ahead using the previous day. All parameters for training and testing the algorithm can be found in the correspondent setup files under \setup. The results for different performance metrics and deep learning algorithms are tabulated in Table 5.

Table 5: Average results for the Tutorial 2 using deep learning approaches.

	R2-Score	RMSE (W)	MAE (W)	MAX (W)
DNN	95.30%	1336.89	887.08	18315.86
LSTM	93.90%	1524.01	1075.18	17875.53
CNN	94.98%	1381.94	995.82	17845.94

As can be seen DNN outperforms all other deep-learning models for all performance metrics, except for the maximum value. However, the maximum value is an artefact from shifting the time series for generating the lagged output. One exemplary output for the forecasted signals is provided below.

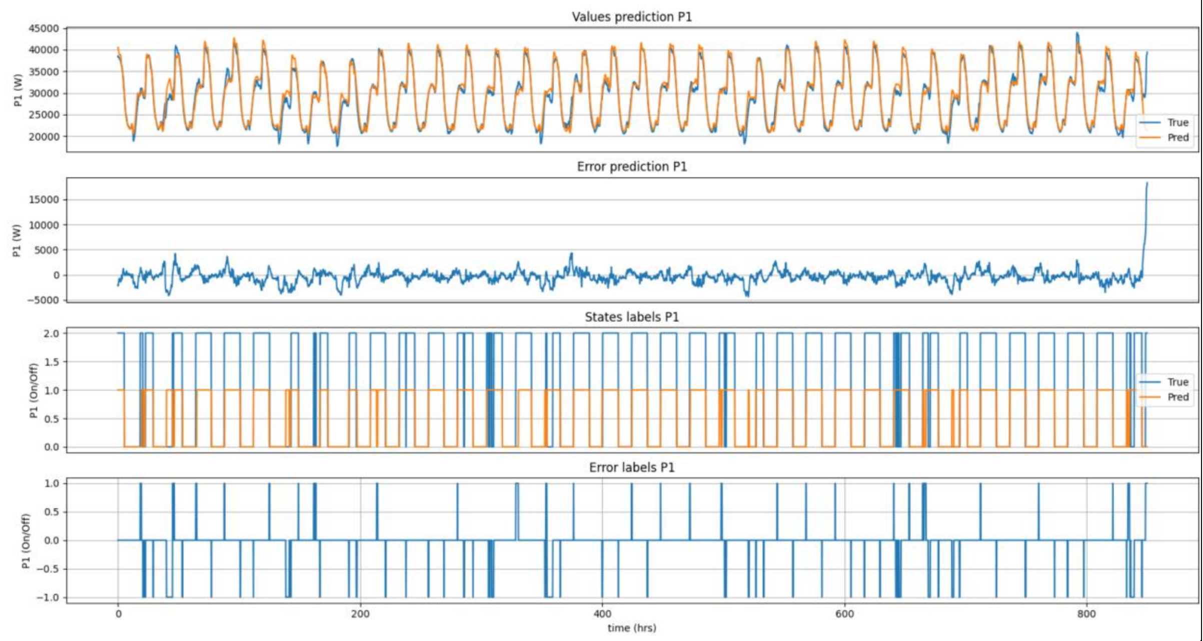


Figure 6: Example output for the load forecasting task.

5.3. Non-Linear Modelling (Tutorial 3)

Non-linear modelling is a task where the relation between input and output values is non-linear, i.e. where the equation is either non-linear in its parameters or variables. A good example can be thermal modelling of power electronics or electric machinery where the fundamental heat conduction equation itself is linear, but non-linearities are introduced through coupling or resistive or semiconductor losses which are themselves a no-linear function of temperature. Fundamentally the temperature on a component can be modelled as follows:

$$\dot{q}(t) = R(T) \cdot I_{rms}^2 \quad (5)$$

where $\dot{q}(t)$ is a time dependent heat source which is generated by a current I_{rms} flowing through a non-linear temperature dependent resistance $R(T)$. The temperature is then calculated using (4):

$$\rho c_p \frac{\partial T}{\partial t} - \nabla \cdot (\mathrm{k}\nabla T) = \dot{q}(t)\varphi(\vec{r}) \quad (6)$$

where ρ is the mass density, c_p the specific heat capacity, and k the thermal conductivity. Furthermore, $\varphi(\vec{r})$ is a spatial function protecting the heat source $\dot{q}(t)$ on the respective volume. Machine learning aims to learn these dependencies in the input and output relation between temperatures and voltage, current, and losses during training. In this tutorial we use the Motor Temperature Dataset from [3], which contains 168 hrs of electric machinery temperature data. The input features are the stator current and voltages, the torque, and the rotational speed, as well as the ambient and the coolant temperature. The output features are three stator temperatures and the magnet temperature of the rotor. All parameters for training and testing the algorithm can be found in the correspondent setup files under \setup. The average results across all four temperatures for different performance metrics and deep learning algorithms are tabulated in Table 6.

Table 6: Average results for the Tutorial 3 using deep learning approaches.

	R2-Score	RMSE (K)	MAE (K)	MAX (K)
DNN	90.89%	3.12	2.57	9.61
LSTM	78.07%	3.81	3.12	9.42
CNN	89.34%	2.71	2.25	7.06

As can be seen CNN outperforms all other deep-learning models except DNNs for the R2-score. One exemplary output for the predicted temperature is provided below.

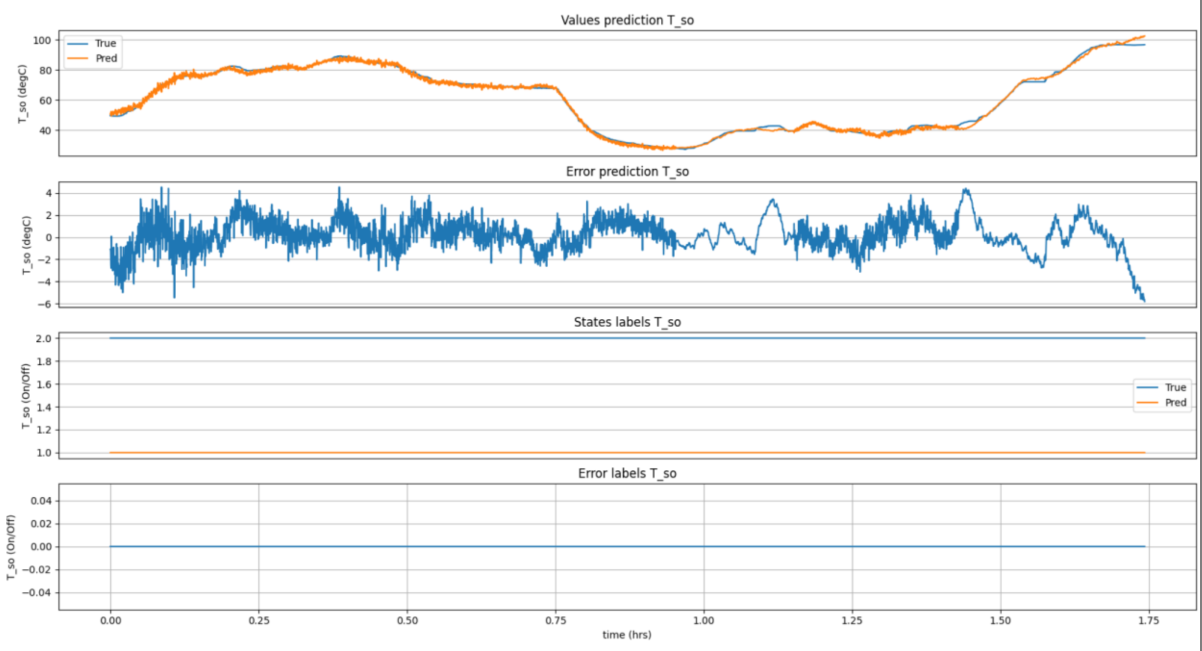


Figure 7: Example output for the non-linear modelling task.

5.4. Anomaly Detection (Tutorial 4)

Anomaly detection describes the task of finding outliers within the data. Often these data are highly unbalanced, i.e. there are much more positive than negative values or vice versa. The aim is to efficiently detect a small number of outliers within large amounts of data. The problem can be mathematically formulated as follows:

$$\hat{y}(t) = \Theta\Big(f\big(x(t)\big)\Big) \tag{7}$$

where $y(t)$ is the status signal, i.e. if a sample at time t is normal or anomalous, $x\,(t)$ are the input signals which provide indication for the status signal, $f\,(\cdot)$ is a function calculating the probability for a sample to be anomalous, and $\Theta(\cdot)$ is a threshold to convert the prediction into a binary variable. In this tutorial we use the Ford Motor Data dataset, which provides vibration inputs, i.e. acceleration values measured on different vehicles, as well as the motor status as binary variable. All parameters for training and testing the algorithm can be found in the correspondent setup files under \setup. Furthermore, since this is a classification approach the setupMdl['loss'] and setupMdl['metric'] variables must be changed to BinaryCrossentropy and accuracy in the mdlPara.py file. The results for different performance metrics and deep learning algorithms are tabulated in Table 7. Unlike the previous approaches this is a classification task, thus we evaluated accuracy and F1 scores.

Table 7: Average results for the Tutorial 4 using deep learning approaches.

	ACC	F1
DNN	74.32%	74.32%
LSTM	52.73%	51.24%
CNN	90.83%	90.83%

As can be seen CNN outperforms all other deep-learning models for all performance metrics. Interestingly LSTM reports significantly worse performance, which is due to the missing temporal relation between consecutive output samples and the strongly uneven distribution of normal and anomalous values. One exemplary output for the predicted state is provided below.

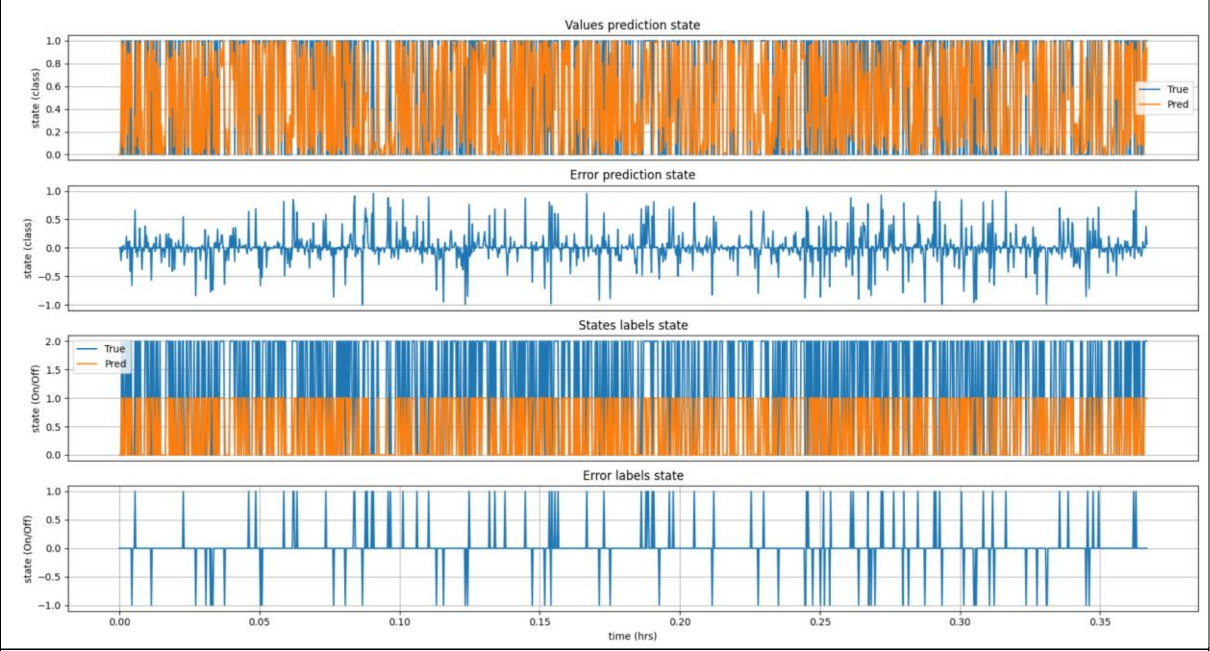


Figure 8: Example output for the anomaly detection task.

5.5. Degradation Modelling (Tutorial 5)

Degradation modelling is a task where a relation between input parameters, time, and slow varying output parameters exists. The aim is to describe the slow varying degradation based on the initial state and the loads applied over time. The problem can be mathematically formulated as follows:

$$y(t) = y_0 + \beta x(t) + \epsilon(t) \tag{8}$$

$$\hat{y}(t) = f\big(y_0, x(t)\big) \tag{7}$$

where $y(t)$ is the degradation signal, $x\,(t)$ are load signals stressing the component, e.g. temperature, and $\epsilon(t)$ is stochastic noise. In this tutorial we will use a battery dataset [5] with measured degradation data (cell capacitance) as output, and temperature, current, and voltage as input features. All parameters for training and testing the algorithm can be found in the correspondent setup files under \setup. The results for different performance metrics and deep learning algorithms are tabulated in Table 8.

Table 8: Average results for the Tutorial 5 using deep learning approaches.

	R2-Score	**RMSE (Ah)**	**MAE (Ah)**	**MAX (Ah)**
DNN	96.44	0.03	0.02	0.30
LSTM	98.54	0.02	0.01	0.08
CNN	98.44	0.02	0.01	0.11

As can be seen LSTM outperforms all other deep-learning models for all performance metrics. One exemplary output for the degradation signal is provided below.

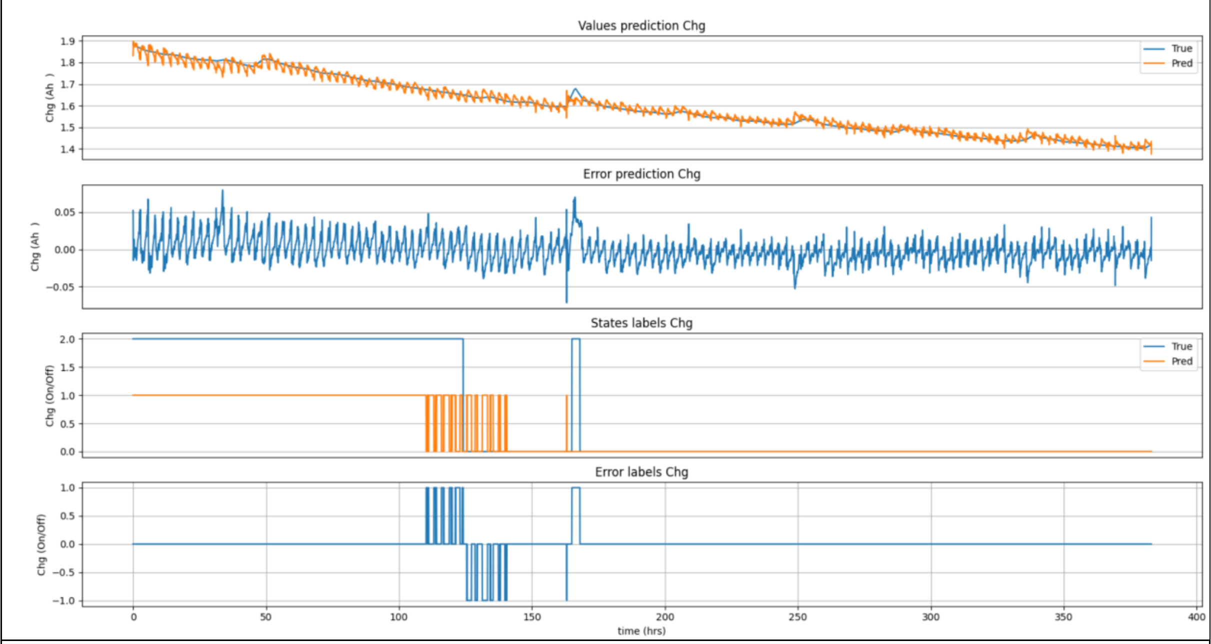


Figure 9: Example output for the degradation modelling task.

5.6. Cross-Domain Modelling (Tutorial 6)

In cross domain modelling the aims is to predict an output value $y(t)$, which is coming from a different domain than the input values $x(t)$ and is mostly weakly correlated or cannot be model using analytical or statistical models. In this tutorial we considered data of driving cycles that have been recorded with a BMW i3 [6]. The dataset consists of several driving cycles and reports numerous features, related to the traction drive system, the battery pack and the coolant and heating system. In this tutorial we try to model the battery current (output) by using typical driving parameters as the input (speed, torque, state of charge, and ambient temperature). All parameters for training and testing the algorithm can be found in the correspondent setup files under \setup. The results for different performance metrics and deep learning algorithms are tabulated in Table 9.

Table 9: Average results for the Tutorial 6 using deep learning approaches.

	R2-Score	**RMSE (A)**	**MAE (A)**	**MAX (A)**
DNN	95.27%	7.43	5.18	109.91
LSTM	89.11%	11.27	8.10	51.59
CNN	78.47%	15.85	9.77	68.14

As can be seen LSTM outperforms all other deep-learning models for all especially for maximum errors. One exemplary output for the estimated current signal is provided below.

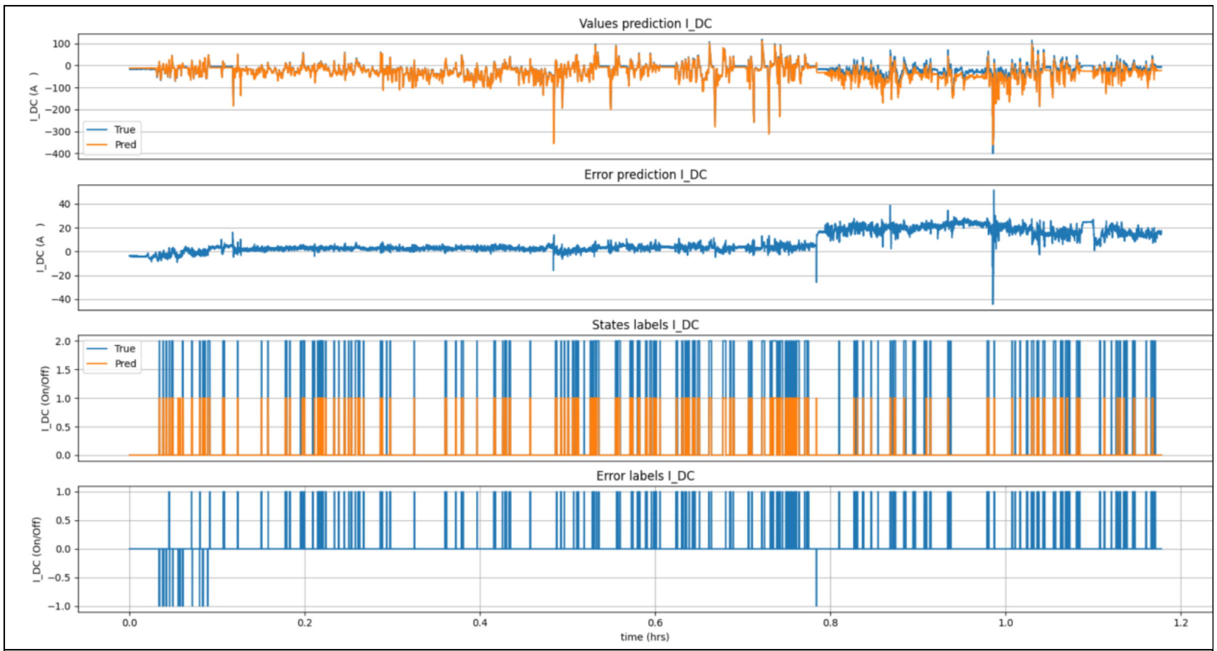


Figure 10: Example output for the cross-domain modelling task.

6. Brief Comparison

To enable comparison with previously published methods two comparisons have been made using the PyDTS toolkit, namely one for the task of energy disaggregation (a highly undetermined problem) and for the task of temperature prediction (a highly non-linear problem). The aim is not to outperform the best performing approaches from the literature, as usually these are very specifically optimised for the respective problem, but to illustrated how good a baseline methodology provided by the PyDTS toolkit will work for the respective tasks. The results are presented in Section 6.1 and Section 6.2.

6.1. Energy Disaggregation

For energy disaggregation we consider again the AMPDs2 dataset from Tutorial 1 and compare the results for a scenario of 10-fold cross validation using the five most used appliances in the dataset, namely 'DWE', 'FRE', 'HPE', 'WOE', and 'CDE'. CNN has been used as it was found to work best for the disaggregation task. The detailed setup files are the same as in the BaseNILM toolkit (https://github.com/pascme05/BaseNILM) and the numerical results are tabulated in Table 10.

Table 10: Detailed results for energy disaggregation evaluating different accuracy metrics and using 10-fold cross validation.

Device	TECA	RMSE (A)	MSE (A)	MAE (A)	MAX (A)
DWE	49.61	0.87	0.93	0.13	6.60
FRE	97.52	0.11	0.33	0.06	2.96
HPE	98.02	0.59	0.77	0.07	47.90
WOE	93.82	0.44	0.66	0.02	29.89
CDE	97.60	0.36	0.60	0.02	39.96
Average	87.31	0.47	0.69	0.06	25.46

For the comparison with the literature three different devices have been considered namely 'HPE', 'WOE', and 'CDE', the results are provided in the Table below.

Table 11: Comparison with previously published results on the same dataset.

REF	Year	Model	TECA	RMSE	MAE
[7]	2021	GAN	-	35.3	-
[8]	2020	GAN	-	38.5	-
[9]	2019	LSTM	-	38.6	-
[10]	2018	CNN	-	63.4	-
PyDTS	2023	CNN	91.22	44.7	2.32

As can be seen PyDTS reaches comparable performances, but obviously cannot come close to NILM specific architectures. One exemplary output for the estimated current signal is provided below.

6.2. Temperature Prediction

For temperature prediction we consider again the Motor Temperature dataset from tutorial 3 and compare the results using the same train, test, and validation split as proposed in [13]. As regression model we have used them same CNN from Tutorial 3, as it has shown the best performance amongst all other models. The detailed setup files can be found under \setup and the numerical results are tabulated in Table 12.

Table 12: Detailed results for motor temperatures evaluating MSE and MAX errors on different testing IDs.

ID	**Time (hrs)**	**Stator Winding**		**Stator Tooth**		**Stator Yoke**		**Permanent Magnet**	
		MSE	**MAX**	**MSE**	**MAX**	**MSE**	**MAX**	**MSE**	**MAX**
60	1.7	4.80	6.73	3.20	5.91	1.66	4.70	30.8	10.9
62	3.3	1.06	6.14	0.62	3.97	0.40	2.92	2.13	5.66
74	3.0	3.42	6.33	2.37	5.66	1.36	3.97	13.6	9.60
Average	8.0	2.74	6.34	1.82	5.02	1.03	3.69	12.5	8.25

As can be seen in Table 12 the difficulty for estimating the temperatures in the different test ID varies significantly, with the lowest errors being found in test ID 62 and the highest in test ID60. On average the results are better for the stator temperatures, which is in line with the input features being mostly stator quantities. To compare the results with the previously published literature a comparison of average errors was made in Table 13.

Table 13: Comparison with previously published results on the same dataset.

REF	Year	Model	MSE (K2)	MAX (K)	Features
[11]	2021	MLP	5.58	14.29	81
[11]	2021	OLS	4.47	9.85	81
[12]	2020	CNN	4.43	15.54	81
[13]	2023	TNN	2.87	6.02	5
PyDTS-B	2023	CNN	4.53	10.91	13
PyDTS-O	2023	CNN	3.56	10.60	13

As can be seen in Table 13 the results obtained from the baseline CNN model (B) implemented in PyDTS are comparable to the results obtained with other machine- or deep-learning architectures. Only physical informed approaches like thermal neural networks [13] perform significantly better.

In the following the abilities of PyDTS for optimizing hyperparameters are exploited to further improve the above results. In detail, the automated grid search is used to tune the input parameters, e.g. the window length of input samples, and hyper-parameter optimization in Keras is used to tune the model parameters respectively. The results for the window length are shown in Figure 11.

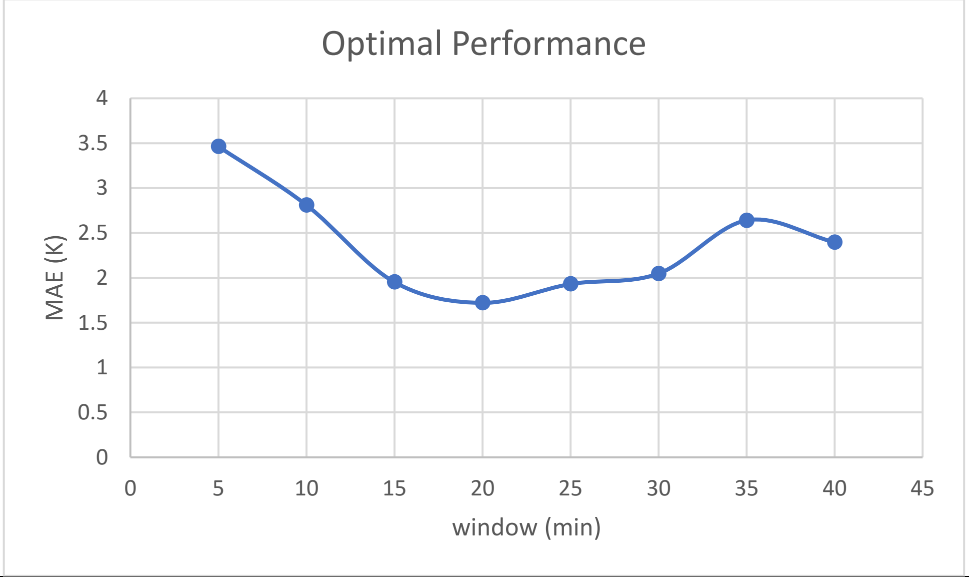
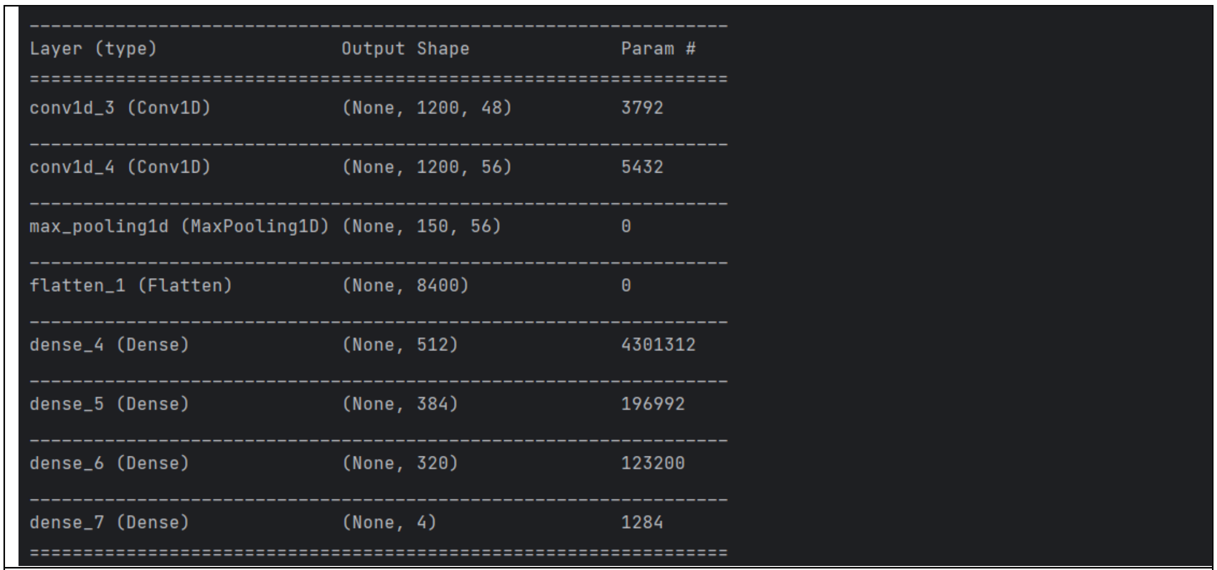


Figure 11: Optimal window length for predicting average motor temperatures.

Using the optimal window length of 20 min, i.e. 1200 samples at 1 Hz sample rate, hyper-parameters of the CNN have been adapted. The results are illustrated in Figure 12 and Figure 13.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_3 (Conv1D)            (None, 1200, 48)          3792
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 1200, 56)          5432
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 150, 56)           0
_________________________________________________________________
flatten_1 (Flatten)          (None, 8400)              0
_________________________________________________________________
dense_4 (Dense)              (None, 512)               4301312
_________________________________________________________________
dense_5 (Dense)              (None, 384)               196992
_________________________________________________________________
dense_6 (Dense)              (None, 320)               123200
_________________________________________________________________
dense_7 (Dense)              (None, 4)                 1284
=================================================================
```

Figure 12: Optimal model structure for predicting average motor temperatures.

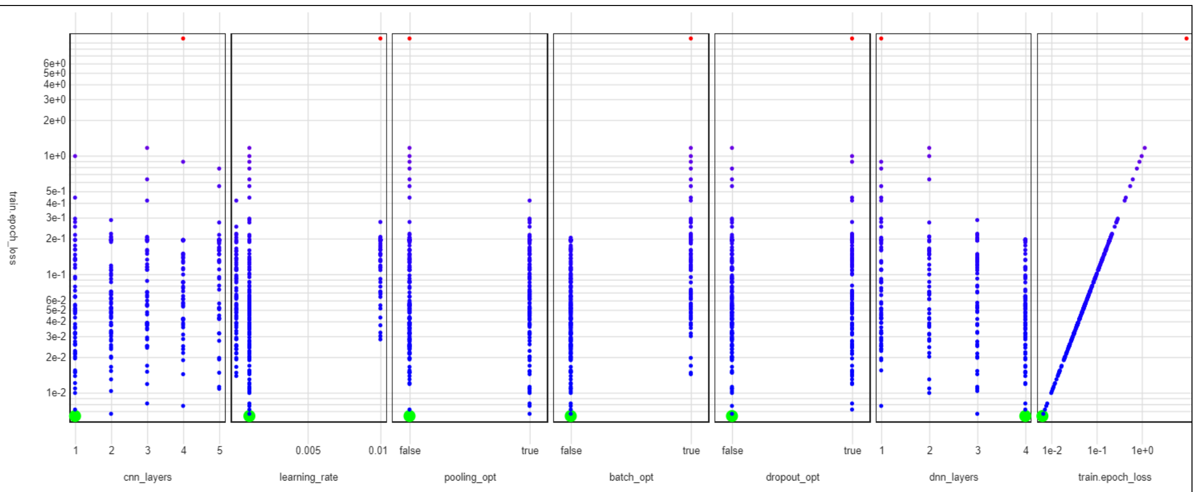


Figure 13: Tensorboard output after model optimisation

In Figure 13 the decisions by the optimizer are clearly reflect, e.g. it can be seen that a pooling layer is favourable or that two CNN layers and three to four DNN layers lead to the best result. The optimisation is leading to an improvement of 3.56 in terms of MSE. It should be mentioned that the optimised model stays still short of reaching performances like a thermal neural network especially when considering maximum and transient error. An example of the prediction is provided below.

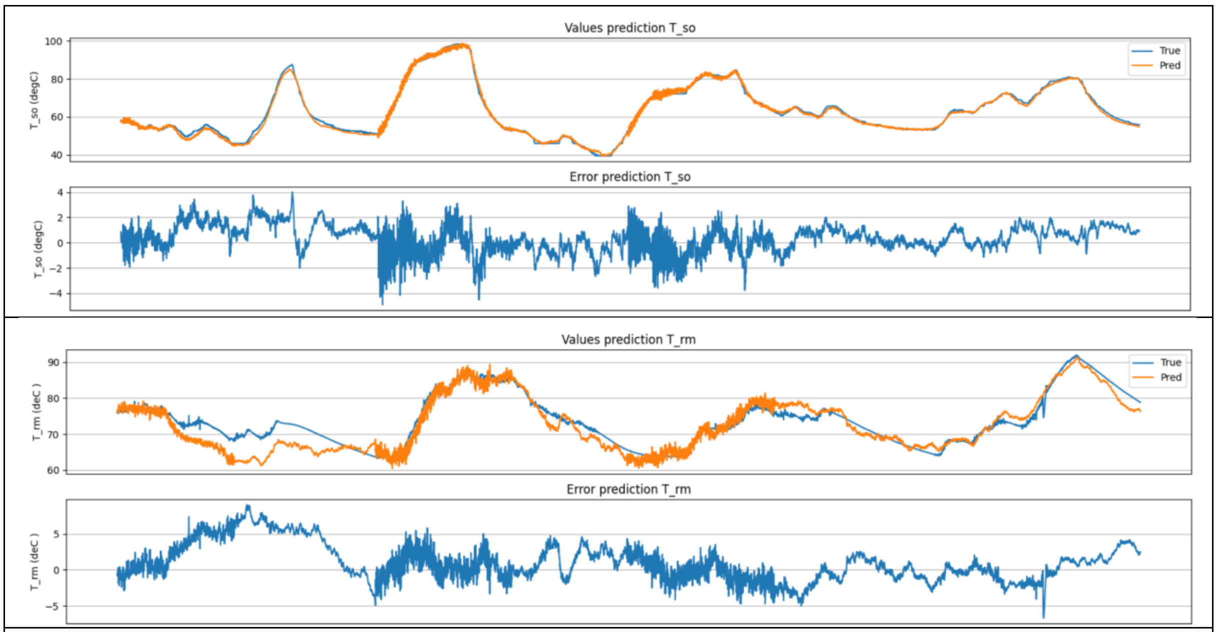


Figure 14: Predicted temperature of the stator yoke and the magnet for ID 74 of the optimised model

7. Conclusion

A python implementation for time-series modelling has been presented. The toolkit can deal with several different problems in the context of time-series modelling and includes different tutorials for trying out these modelling techniques. The toolkit cannot achieve the performance of specific toolkits for the respective modelling approaches, but it can help to gain understanding and serve as a baseline system.

8. References

[1] Makonin, S., Ellert, B., Bajić, I. et al. Electricity, water, and natural gas consumption of a residential house in Canada from 2012 to 2014. Sci Data 3, 160037 (2016). https://doi.org/10.1038/sdata.2016.37

[2] Dedesoriano. (August 2022). Electric Power Consumption. Retrieved from https://www.kaggle.com/datasets/fedesoriano/electric-power-consumption

[3] W. Kirchgässner, O. Wallscheid and J. Böcker, "Estimating Electric Motor Temperatures With Deep Residual Machine Learning," in IEEE Transactions on Power Electronics, vol. 36, no. 7, pp. 7480-7488, July 2021, doi: 10.1109/TPEL.2020.3045596

[4] Wichard, Joerg D. "Classification of Ford Motor Data." Computer Science (2008).

[5] Bills, A., Sripad, S., Fredericks, L. et al. A battery dataset for electric vertical takeoff and landing aircraft. Sci Data 10, 344 (2023). https://doi.org/10.1038/s41597-023-02180-5

[6] Matthias Steinstraeter, Johannes Buberger, Dimitar Trifonov, October 19, 2020, "Battery and Heating Data in Real Driving Cycles", IEEE Dataport, doi: https://dx.doi.org/10.21227/6jr9-5235

[7] M. Kaselimi, N. Doulamis, A. Voulodimos, A. Doulamis and E. Protopapadakis, "EnerGAN++: A Generative Adversarial Gated Recurrent Network for Robust Energy Disaggregation," in IEEE Open Journal of Signal Processing, vol. 2, pp. 1-16, 2021, doi: 10.1109/OJSP.2020.3045829

[8] M. Kaselimi, A. Voulodimos, E. Protopapadakis, N. Doulamis and A. Doulamis, "EnerGAN: A GENERATIVE ADVERSARIAL NETWORK FOR ENERGY DISAGGREGATION," ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 2020, pp. 1578-1582, doi: 10.1109/ICASSP40776.2020.9054342

[9] M. Kaselimi, N. Doulamis, A. Doulamis, A. Voulodimos and E. Protopapadakis, "Bayesian-optimized Bidirectional LSTM Regression Model for Non-intrusive Load Monitoring," ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, UK, 2019, pp. 2747-2751, doi: 10.1109/ICASSP.2019.8683110

[10] Chen, Kunjin, et al. "Convolutional sequence to sequence non-intrusive load monitoring." the Journal of Engineering 2018.17 (2018): 1860-1864.

[11] Kirchgässner, Wilhelm, Oliver Wallscheid, and Joachim Böcker. "Data-driven permanent magnet temperature estimation in synchronous motors with supervised machine learning: A benchmark." IEEE Transactions on Energy Conversion 36.3 (2021): 2059-2067.

[12] W. Kirchgässner, O. Wallscheid and J. Böcker, "Estimating Electric Motor Temperatures With Deep Residual Machine Learning," in IEEE Transactions on Power Electronics, vol. 36, no. 7, pp. 7480-7488, July 2021, doi: 10.1109/TPEL.2020.3045596

[13] Kirchgässner, Wilhelm, Oliver Wallscheid, and Joachim Böcker. "Thermal neural networks: Lumped-parameter thermal modeling with state-space machine learning." Engineering Applications of Artificial Intelligence 117 (2023): 105537.