

Introduction to Machine Learning — Assignment 2

Predict the Season of the Year by Power Consumption in Denmark



1. Introduction

You have learned how convolutional neural networks (CNNs) can be used for **image classification**. In this assignment, you will see that CNNs can also be effective for **time-series classification**, where data points are recorded at consecutive time intervals. Specifically, you will implement **three different neural network architectures**:

1. A **fully-connected (MLP) network**.
2. A **1D convolutional neural network (1D-CNN)** directly on time-series data.
3. A **2D convolutional neural network (2D-CNN)** using a time-series-to-image transformation.

Your ultimate goal is to predict which **season of the year** (winter, spring, summer, or autumn) corresponds to a **24-hour record of power consumption and production** in Denmark, drawn from the **Open Power System Data** dataset.

To maximize learning, you will handle **raw, unfiltered data**—which means you must perform data cleaning, preprocessing, splitting, and engineering. This ensures familiarity with the entire data science pipeline from raw inputs to final model.

2. Dataset

We provide a subset of the **Open Power System Data**, which contains various hourly power-related measurements (e.g., consumption, wind generation, solar capacity) across EU countries from 2015 to approximately 2020. You should focus on Denmark data and extract **3 features** for this country (all in MW):

1. Total power load;
2. Wind generation;
3. Solar generation.

Note: be careful with the countries abbreviations, Denmark is not DE!

2.1. Data Files

- **opsd_raw.csv**: A raw CSV file with many columns (time, country, load, generation, etc.).
- **README.md**: Contains descriptions of each column. Use them to identify feature columns you should extract.

Note: some columns are irrelevant. Part of the challenge is to identify which column truly corresponds to Denmark's load and green energy production, extract it, and ignore the rest.

2.2. Data Preprocessing Requirements

1. **Identify & Extract**: Locate the column for **Denmark's total power load and wind&solar generation**.
2. **Create 24-Hour Records**: Aggregate the data so that each “instance” (row) in your final dataset represents a single day's worth of hourly measurements—i.e. **3** arrays of length 24 (power load, wind generation, solar generation).
3. **Label Each Day**: Determine which season each 24-hour record belongs to. The seasons can be defined in a straightforward calendar-based way (e.g., March 1–May 31 as spring, etc.).
4. **Handle Missing Data**: If any rows or columns are incomplete, decide how to handle them (imputation, removal, etc.). Document your approach.

Important: You must show the exact steps of your data preprocessing in your code. Simply providing a final “clean” dataset is insufficient.

3. Tasks and Deliverables

Task 1: Exploratory Data Analysis (EDA) and Preprocessing (25 points)

1. **Load the Data**
 - Read `opsd_raw.csv` into a `DataFrame`. Identify the relevant columns.

- **Manually confirm** which column are for Denmark's power load and productions by referencing `README.md`.
2. **Data Inspection**
 - Print the shape (rows $\times$ columns) of the raw data and the first few lines.
 - Check for **missing values** (NaNs). Decide how you'll handle them (drop rows, fill forward, etc.).
 3. **Form 24-Hour Arrays**
 - Convert the hourly data into daily slices of length 24.
 - Ensure each daily record lines up with a single date (e.g., from midnight to midnight).
 - Show at least 5 sample arrays to confirm correctness.
 4. **Label Seasons.** Map each day's date to a season label. Summarize how many samples belong to each season.
 5. **Train-test split.** Split the daily records into train (70%), validation (15%), and test (15%) sets.
 6. **Data scaling.** Apply one of the standardisation (or scaling) methods you've learnt on the IML course on every feature column. Avoid data leakage when fit a scaler.
 7. **Brief Analysis**
 - Plot at least one example of a daily consumption profile for each season.
 - Include 1–2 **personal observations** (e.g., “We see that winter consumption is generally higher than summer.”). This helps ensure you're truly analyzing the data beyond a purely automated approach.

Deliverables for Task 1

- Data preprocessing code.
- A short textual explanation (one paragraph) of your preprocessing approach, referencing the dataset README.
- Plots or tables that illustrate daily consumption patterns and distribution across seasons.
- Train, validation and test datasets with the standardised values.

Task 2: Baseline MLP (Fully-Connected Network) (15 points)

1. **Implement an MLP in PyTorch** with at least:
 - One hidden layer (e.g., 32–128 neurons).
 - Non-linear activation (e.g., ReLU).
2. **Input Representation**
 - Flatten the 24-hour sequence into a 24-dimensional vector.
 - Normalize or standardize the inputs if needed.

3. Train & Evaluate

- Show training curves for loss and accuracy.
- Evaluate on both validation and test sets, and report final accuracy.

Important

Provide **inline comments** explaining why you picked certain hyperparameters (learning rate, hidden size). Submit a short training log with epoch numbers, training loss, and validation accuracy.

Deliverables for Task 2

- PyTorch code for building and training the MLP.
 - Final “test set” predictions saved or displayed.
 - A confusion matrix or classification report comparing predicted vs. actual seasons on the test set.
-

Task 3: 1D-CNN on Raw Time-Series (20 points)

1. 1D Convolution Architecture

- Use PyTorch `Conv1d` layers to process 3-channels sequences of length 24.
- At least one convolutional layer, one pooling layer, and a final fully connected layer for classification.

2. Hyperparameter Tuning

- Try different kernel sizes (e.g., 3 vs. 5).
- Consider optional regularization (dropout, batch norm).

3. Comparison vs. MLP

- Report the validation/test accuracy.
- Summarize any performance gains (or losses) vs. the MLP approach.

Important:

In top of each layer in `__init__()` **add a comment** with calculated input and output shapes of tensor for this layer. For example:

```
...
# (batch_size, 3, 24) -> (batch_size, 32, 22)
self.conv1 = nn.Conv1d(3, 32, kernel_size=2)
# (batch_size, 32, 22) -> (batch_size, 64, 18)
...
```

This ensures you conceptually understand the structure, rather than just letting an AI tool generate code.

Deliverables for Task 3

- Same as for the previous task
-

Task 4: 2D Transform & 2D-CNN (20 points)

You will now transform each 24-hour time-series into a **2D “image”** before feeding it into a 2D CNN.

1. Choose a transformation from the possible options:

- **Option A:** Use `pyts` (e.g., `GramianAngularField`, `RecurrencePlot`, or `MarkovTransitionField`).
- **Option B:** Create a simple matrix that arranges the data in a 2D pattern (though 24 points is small, you might artificially reshape it into 6×4 or some other arrangement).
- **Option C:** Another custom transformation that you can justify.

Note that you need to apply the transformation on every feature from the original time series. Thus, at the end of the day, every day record should have a shape $3 \times \text{Width} \times \text{Height}$, where *Width* and *Height* are from 2D transform. Consider total load, wind and solar production features as channels in RGB image.

2. Train a 2D CNN

- Use standard 2D convolutions (`Conv2d` in PyTorch).
- At least one pooling layer, at least one hidden conv layer, and a final fully-connected block.

3. Evaluate & Discuss

- Compare your 2D CNN’s performance to both the MLP and the 1D-CNN.
- In a short paragraph, discuss whether the 2D transform improved accuracy, and propose at least one reason why it did (or did not).

Important

Provide your own explanation of the chosen 2D transform and why it might help. Reference one piece of reading or a documentation snippet from `pyts` or another library, citing it properly.

Deliverables for Task 4

- Same as for the previous task
-

4. Submission Format & Grading

1. Notebook/Code

- A `main.ipynb` file containing **all** code. We should be able to run it end-to-end.
- `requirements.txt` listing the packages needed to run the script
- Each model must have a function (or code block) that outputs predictions given a test set.

2. Short Report (1–3 pages) (20 points)

- Summaries for each task, key plots/tables, confusion matrices, accuracy results, and your personal insights.
- Highlight how you handled data splits, transformations, and hyperparameters.
- For the chosen 2D transform(s), cite at least one reference or library documentation. **Explain** in details how the selected method transfer a sequence into the matrix.
- Report format must be `.pdf` or `.word`

Very important note: please **do not change the proposed structure** of the Assignment, otherwise we could reduce a noticeable amount of points for unclear solution.

5. Advice for Avoiding Over-Reliance on AI Tools

- **Explain Your Reasoning:** In code comments and your report, clearly state *why* you chose certain hyperparameters and transformations. AI-generated code often lacks domain-context justifications.
 - **Cite the Dataset README:** For extracting the feature columns, show the exact line from the README that indicates it's Denmark's load. Summarize that in your own words.
 - **Reflect on Data Splitting:** Provide 2–3 original sentences explaining your approach to preventing data leakage across different seasons. This real-world perspective is hard for an AI to replicate without direct instruction.
-

6. Submission Instructions

- **Deadline: 13.04.2025, 23:50**
- **Upload** your notebook/script, final **short report (PDF)**, and any additional files (images, etc.) to Moodle.

- Ensure your code runs end-to-end. We will do a fresh environment install and run your `.ipynb` directly.

Late Policy: 10% penalty per day late, up to 3 days. After that, submissions may not be accepted, except legal excuse from DoE, but make sure you've notified me (tg @GRoman20) BEFORE the deadline.

7. Useful sources

[1] A Python Package for Time Series Classification (for 2d transformations) - <https://pyts.readthedocs.io/en/stable/>

[2] Spectrogram, as alternative way for 2d transformation - <https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.spectrogram.html>

[3] Dataset: https://data.open-power-system-data.org/time_series/2020-10-06