ML assignment 2

Ruslan Gatiatullin

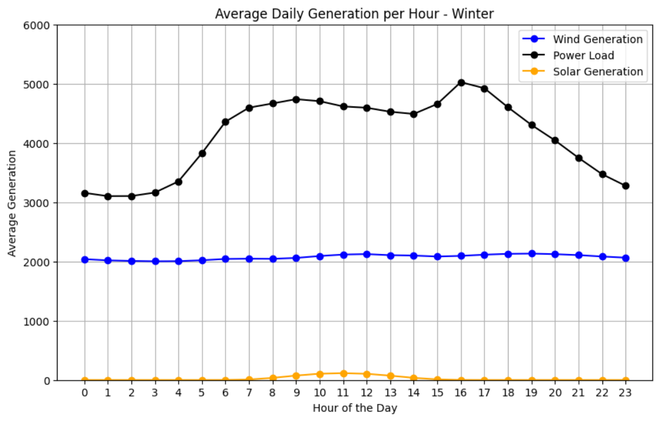
April 18, 2025

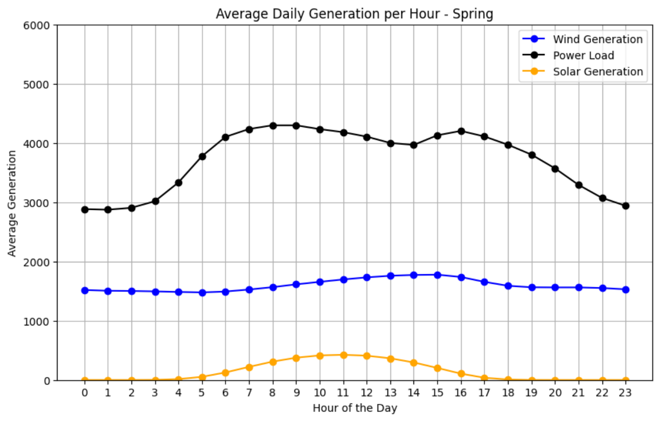
Task 1

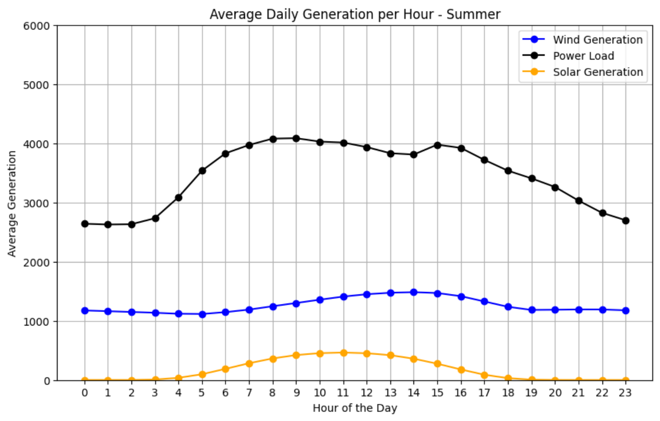
As we can see from dataset README file, we need the following columns:

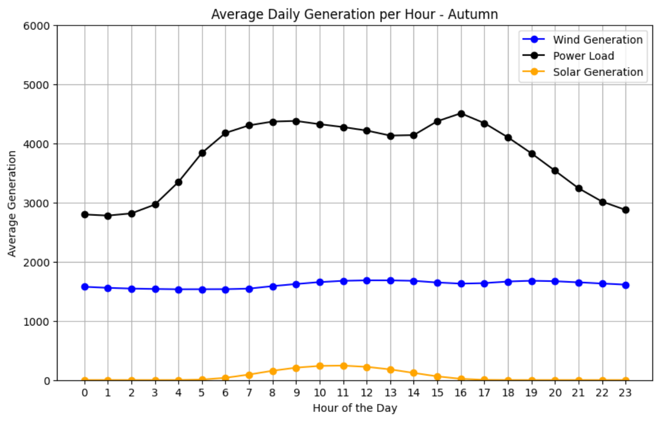
- DK_load_actual_entsoe_transparency - Total load in Denmark in MW as published on ENTSO-E Transparency Platform
- DK_solar_generation_actual - Actual solar generation in Denmark in MW
- DK_wind_generation_actual - Actual wind generation in Denmark in MW

Firstly, I dropped all other columns, then I needed to handle missing values. My approach was to assign the closest not NaN value to them, since they will typically be similar (and the dataset doesn't have long intervals of just NaN's). Then I created 3 separate dataframes with 24 hour records because it was easier to work with than 1 big dataframe with 3 dimensions. After this, I applied MinMaxScaler, assigned seasons to each day record, concatenated 3 dataframes into 1, and did split into train, test, and validation sets.







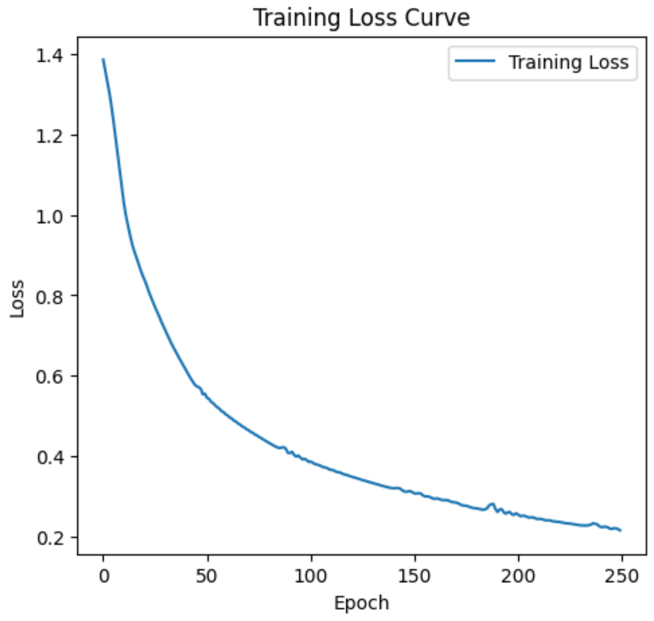


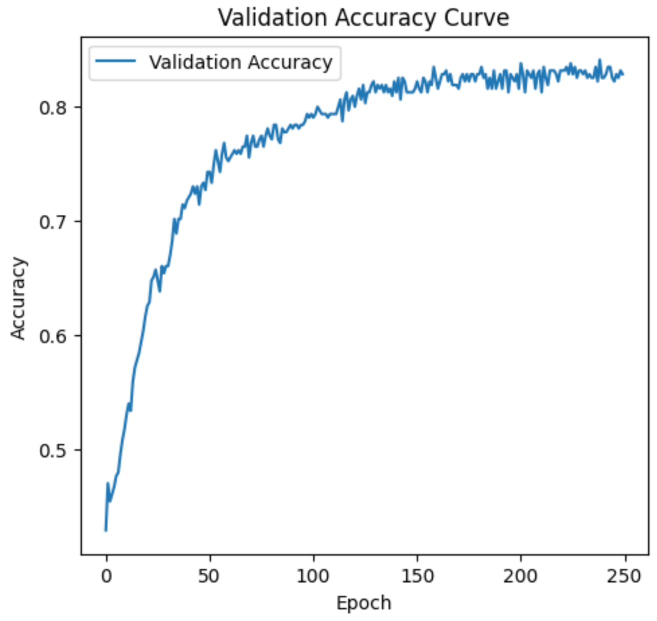
Observations:

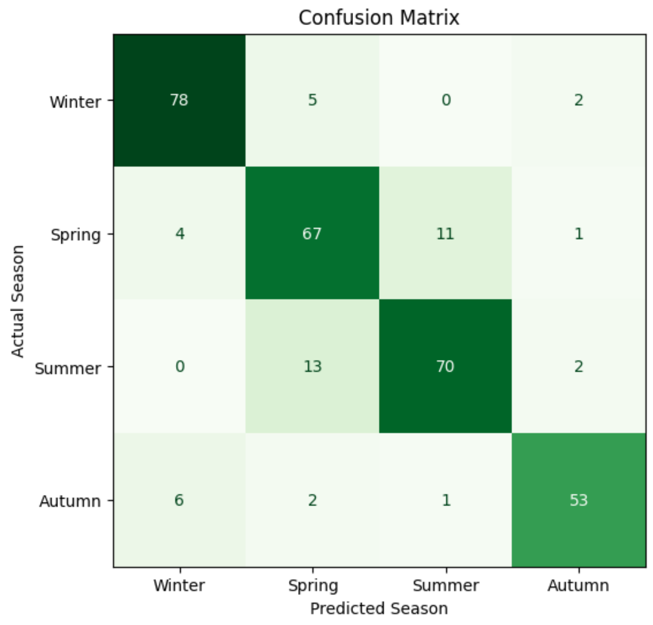
1. Wind generation is almost constant throughout 1 season
2. Solar electricity is generated between 4 am and 6 pm and is very different for each season

Task 2

Final Test Accuracy: 0.8508

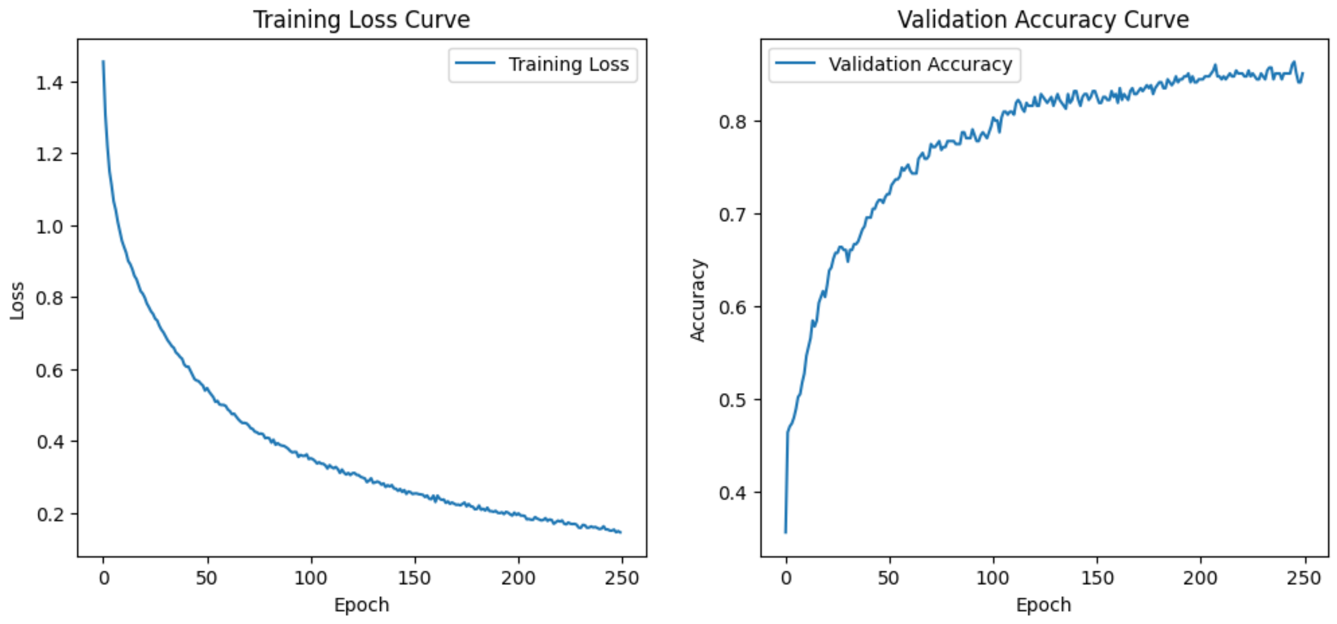


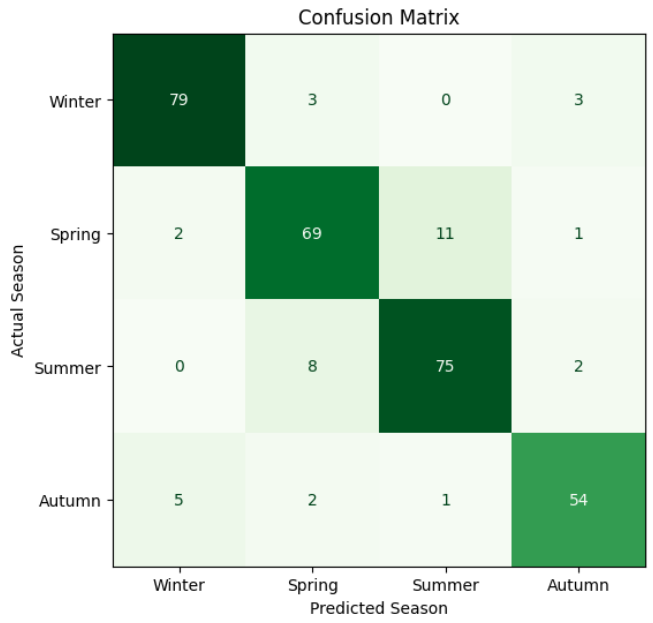




Task 3

Final Test Accuracy: 0.8794



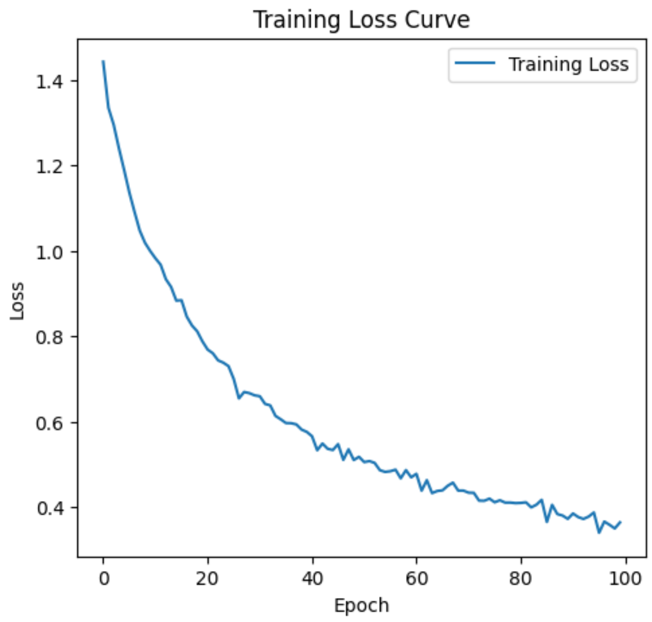


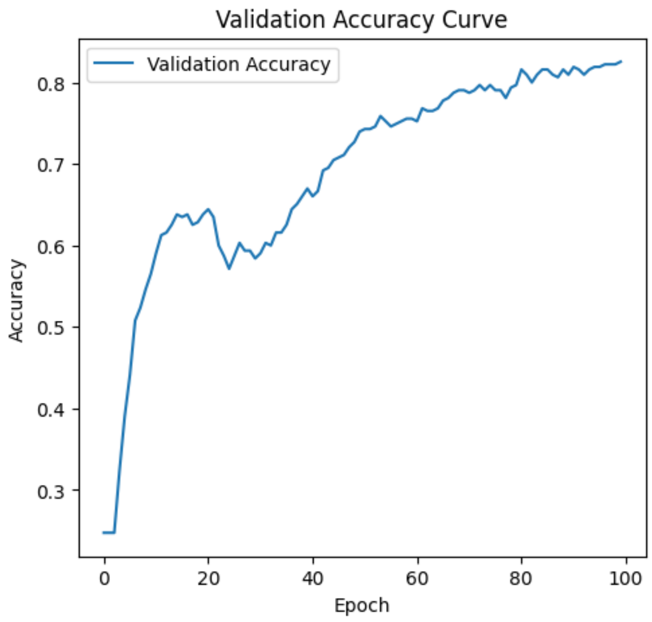
Actual \ Predicted	Winter	Spring	Summer	Autumn
Winter	79	3	0	3
Spring	2	69	11	1
Summer	0	8	75	2
Autumn	5	2	1	54

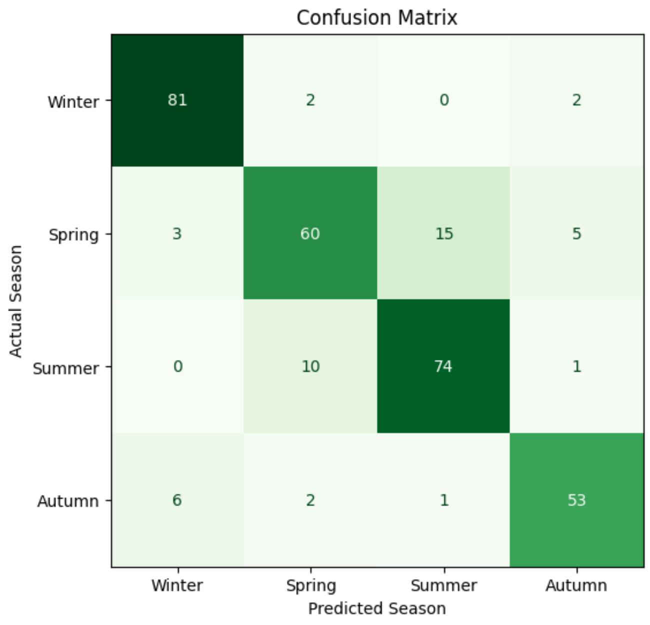
1D CNN has a silightly better performance than MLP. One reason for this is that we're using batch normalization and dropout. The first helps to stabilize and accelerate the training by reducing internal covariate shift. The latter prevents overfitting by randomly disregarding some neurons.

Task 4

Final Test Accuracy: 0.8508







	Winter	Spring	Summer	Autumn
Winter	81	2	0	2
Spring	3	60	15	5
Summer	0	10	74	1
Autumn	6	2	1	53

2D CNN has exactly the same performance as MLP and a slightly worse one than 1D CNN. It is mainly because our image consists of only 24 pixels, while usually pictures would have 1000 times more. Additionally, 1D CNNs are designed to capture patterns in sequential data, such as time-series, by applying convolution along the temporal axis. For simpler tasks or when the data is already well-structured an MLP might perform better because it directly learns from the raw features without relying on spatial relationships.

Data splits, transformations, and hyperparameters

Data was split into train, test, and validation sets using train_test_split from sklearn with a seed 52. All 3 sets don't overlap, don't contain features derived from the season, have the same transformations. This ensures no data leakage.

GramianAngularField from pyts was used for as the transformation mainly because it provides scaling to the data, *which is desirable for CNN inputs*. It also gave way better results compared to RecurrencePlot and MarkovTransitionField. Although MinMaxScaler had to be used for best results, which decreased performance of MLP and 1D CNN by a few percentage points

Hyperparameters were adjusted manually.

Gramian Angular Field explanation

Key steps of Gramian Angular Field are as follows

- Normalization. Scale the time series to values $\in$ [-1, 1] to represent values as angles.
- Go to polar coordinates. Treat normalized values as angles (θ) in polar coordinates. Radius = timestamp, angle = $\arccos(x_i)$, where x_i is the normalized value.
- Compute pairwise trigonometric differences or sums (differences in our case) between angles. Sum: $\cos(\theta_i + \theta_j)$ - Captures temporal correlation.

  Difference: $\sin(\theta_i - \theta_j)$ - Captures variability.
- The resulting matrix will be symmetric and will encode temporal dynamics in a 2D structure.