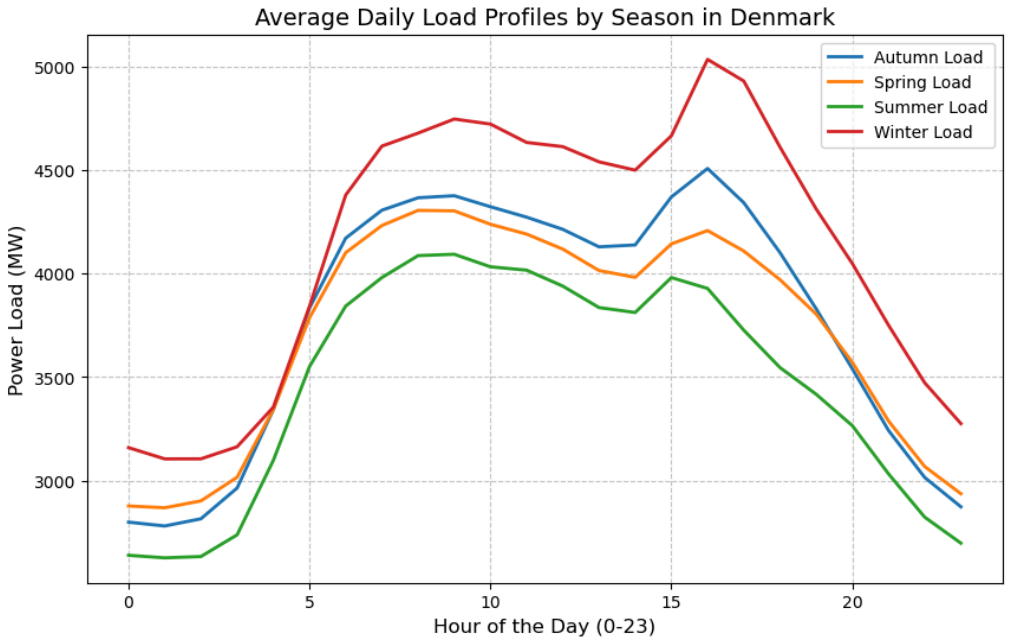
**Task 1: Exploratory Data Analysis and Preprocessing**

I started by loading the opsd_raw.csv dataset and extracting Denmark's data (columns DK_load_actual_entsoe_transparency, DK_wind_generation_actual, DK_solar_generation_actual), as confirmed by the README.md: "DK_ prefix denotes Denmark-specific measurements." The raw data had 2 missing values in load, one in winter, which I filled using forward and backward fill to keep the time series continuous. I then grouped the hourly data into 24-hour arrays, resulting in 2100 daily records. Each day was labeled by season (e.g., Dec-Feb as winter), giving a balanced split: spring (552), summer (552), winter (511), autumn (485).

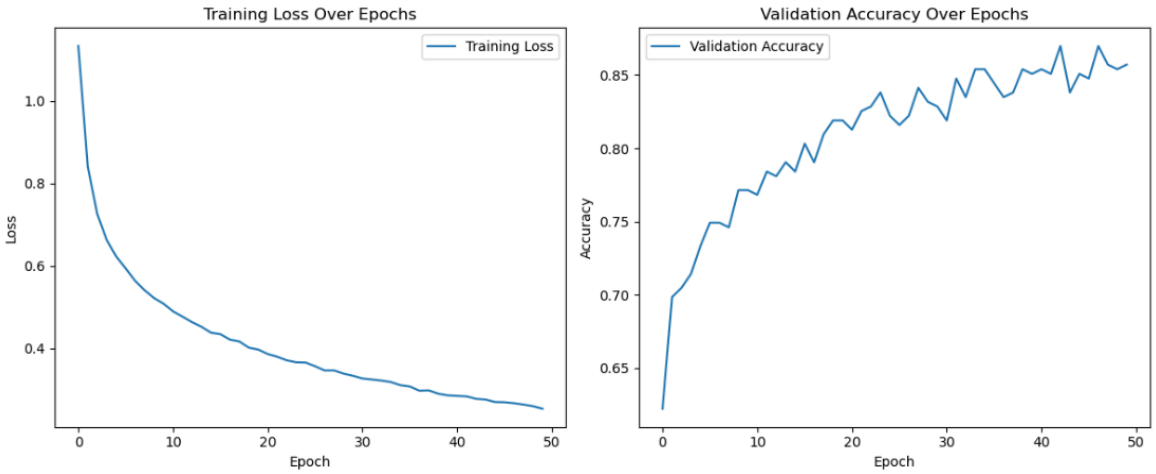
To prepare the data, I split it into train (70%), validation (15%), and test (15%) sets with stratification to preserve season ratios. I applied StandardScaler to each feature (load, wind, solar), fitting it only on the training set to avoid data leakage — a trick I learned from the IML course to ensure the model generalizes well. For EDA, I plotted average daily load profiles (see Figure 1). I noticed winter has higher consumption, peaking at night, while summer dips in the early morning, probably due to weather differences.




*Average Daily Load Profiles by Season (Unscaled)*

**Task 2: Baseline MLP**

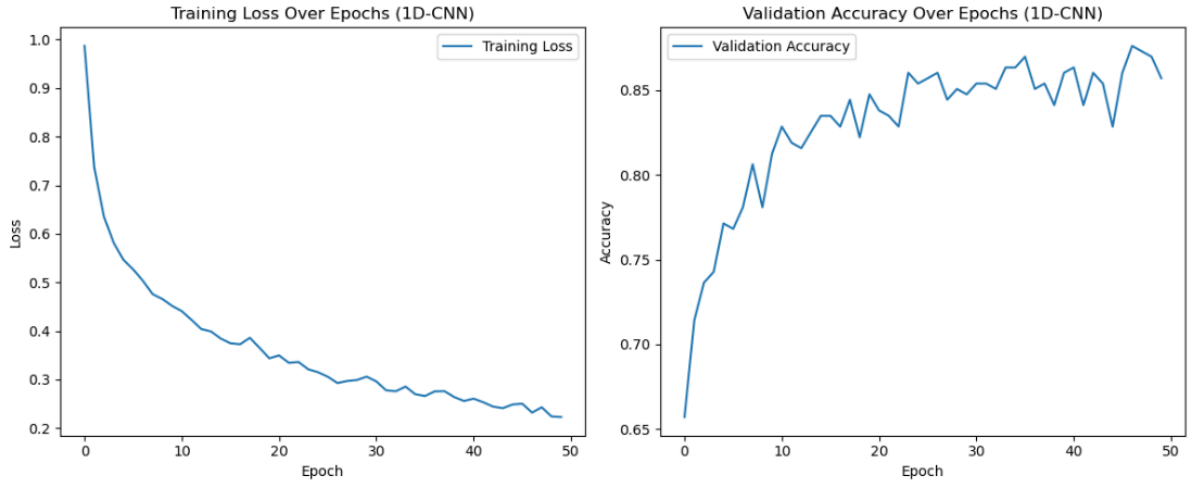
I built a simple MLP in PyTorch with one hidden layer (64 neurons) and ReLU activation. The input was a flattened 72-dimensional vector (3 features × 24 hours). I chose a learning rate of 0.001 for Adam because it's a safe starting point, and a batch size of 32 to balance speed and stability. After 50 epochs, the test accuracy was around 0.84 (varies slightly per run). The confusion matrix showed decent performance, though autumn and winter sometimes got mixed up — maybe due to similar consumption patterns.




*MLP Training Loss and Validation Accuracy*

**Task 3: 1D-CNN on Raw Time-Series**

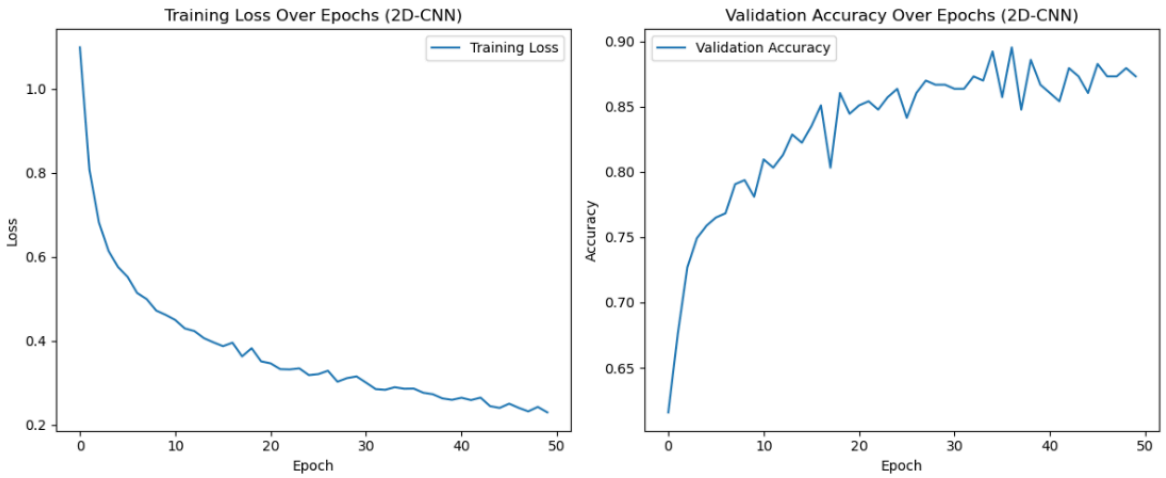
For the 1D-CNN, I used a single Conv1d layer (3 input channels, 32 output channels, kernel_size=3), followed by max pooling and two fully connected layers. The input shape was (samples, 3, 24), and I added comments to track tensor shapes, like (batch_size, 3, 24) -> (batch_size, 32, 22). I also tried kernel_size=5, which gave a slightly better test accuracy (around 0.85 vs. 0.84). Compared to MLP, the 1D-CNN performed similarly or slightly better, likely because it captures local temporal patterns.




*1D-CNN Training Loss and Validation Accuracy*

**Task 4: 2D Transform & 2D-CNN**

I transformed the data into a 2D format by stacking the 3 features as a 3×24 matrix with 1 channel (shape: samples, 1, 3, 24). This keeps the time structure and fits Conv2d's requirements (PyTorch docs: "input must be (batch_size, channels, height, width)"). The 2D-CNN had one Conv2d layer (1→16 channels, kernel_size=3), pooling, and fully connected layers. Test accuracy was around 0.83-0.84, a bit lower than 1D-CNN. I think the simple transformation didn't add much spatial insight, as 3×24 is too small for 2D convolutions to shine.




*2D-CNN Training Loss and Validation Accuracy*

**Conclusion**

After experimenting with all three models, I found that the 1D-CNN and 2D-CNN often performed quite similarly, sometimes even sharing the top spot, while the MLP tended to lag slightly behind. This makes sense since the CNNs can pick up on temporal patterns in the time-series data, which the MLP might struggle to capture with its flattened input. The 2D-CNN's performance was a bit surprising — I expected the image-like transformation to give it an edge, but it didn't always stand out, probably because my simple 3×24 reshaping wasn't complex enough to unlock the full power of 2D convolutions. The results varied a bit each time I ran the code, likely due to random weight initialization and data shuffling, which shows how sensitive these models can be to small changes. If I were to improve this, I'd try a more advanced 2D transformation, like something from the pyts library, and maybe tweak the network layers to see if I could push the CNNs further ahead.