AISE 4010 Project Demo

Time Series Trends and Forecasting of Ontario Electricity Demand

By: Apon Das, Rayyan, Ritwick Vemula

Motivation & Project Goal

Did you know Ontario is Canada’s **most populous province**, with over 16 million residents, and two out of every 5 Canadians lives in Ontario. Accelerating at **3.1%** increase (2022-2023).

Motivation & Project Goal

•Electricity demand in Ontario fluctuates hourly and seasonally.

•Accurate short-term forecasting supports grid stability and cost-efficient operations.

•Avoids over-generation, under-supply, and unnecessary reserve usage.

•Increasing grid complexity requires more advanced, data-driven forecasting methods.

Project Goals:

- Develop short-term (1-hour-ahead) electricity demand forecasts.

Compare LSTM, GRU, TCN, FNN, AND bidirectional deep learning models.

Use engineered time-series features to capture daily and weekly patterns

Models and Training

Dataset: **IESO Ontario Demand (Kaggle)** — hourly data for 2025.

Columns: Date, Hour, Market Demand, Ontario Demand.

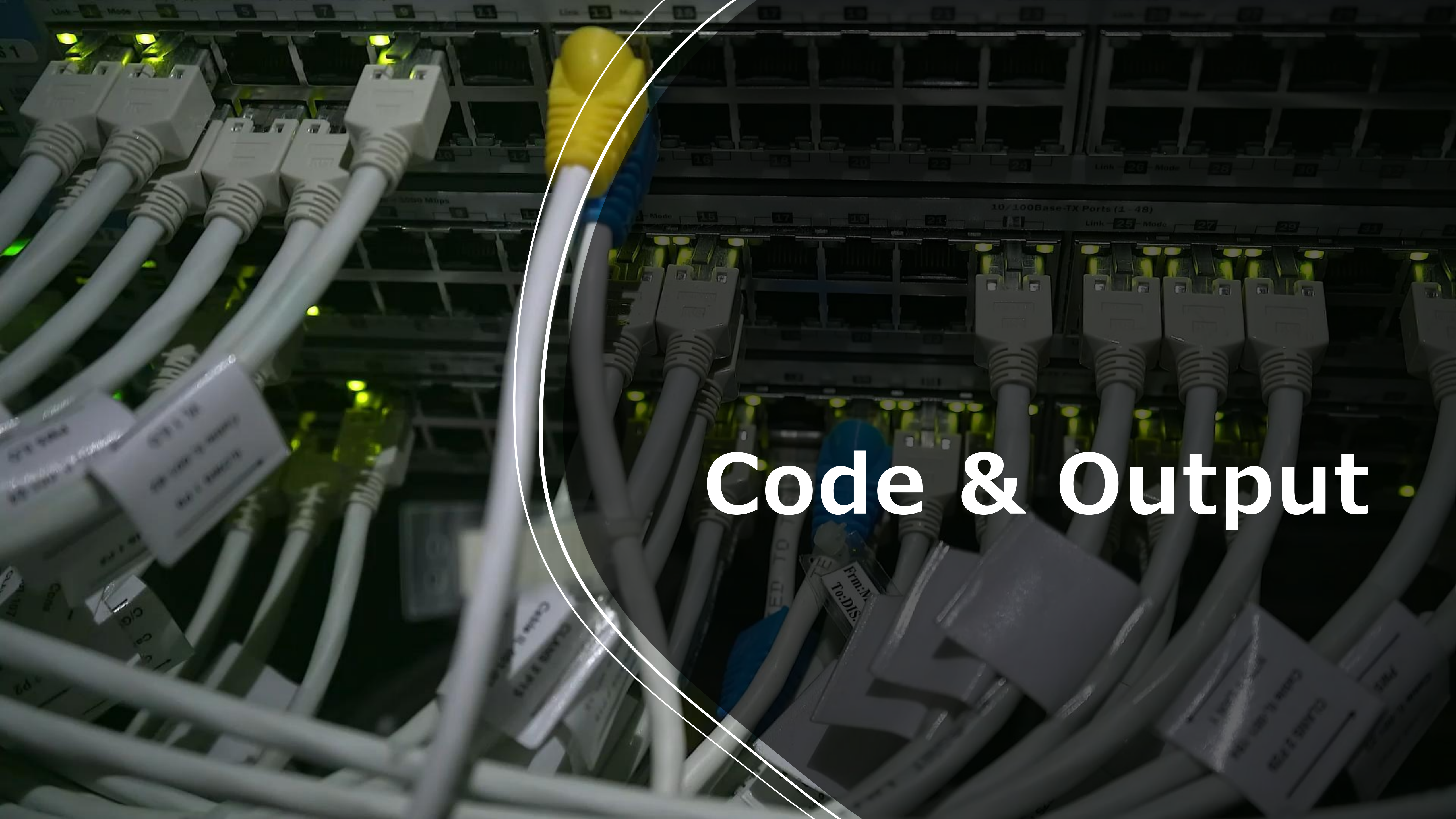
Preprocessing: Date–Hour merged, Hour 24 corrected, datetime index.

Train: Jan–June • Test: July–Sept

Models:

-LSTM, GRU, TCN, FNN

Bidirectional LSTM



Code & Output

LSTM

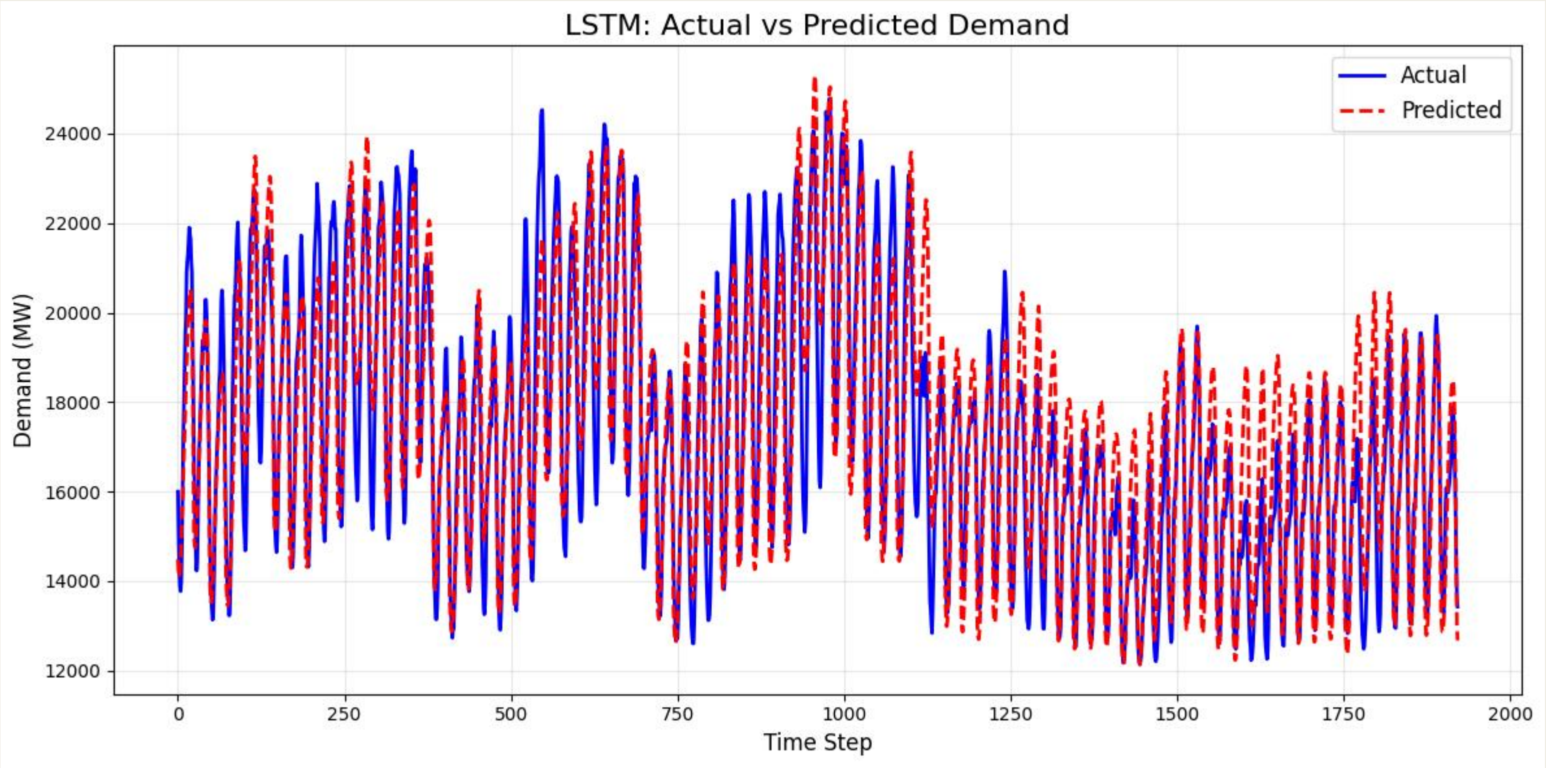
Since the dataset is looking at the Short-Term Dataset and predicting the consumption we want our model train on parameters that won't cause it over perform

Parameters:

- 50 neurons
- 0.2 dropout rate
- Adam's learning rate of 0.001

```
def create_lstm_model(input_shape, neurons=50, dropout=0.2):
    model = Sequential([
        LSTM(neurons, activation='relu', input_shape=input_shape,
             return_sequences=True),
        Dropout(dropout),
        LSTM(neurons, activation='relu'),
        Dropout(dropout),
        Dense(1)
    ])
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')
    return model
```

LSTM



LSTM: Actual vs Predicted Demand
Actual
Predicted
Demand (MW)
24000
22000
20000
18000
16000
14000
12000
0
250
500
750
1000
1250
1500
1750
2000
Time Step

GRU

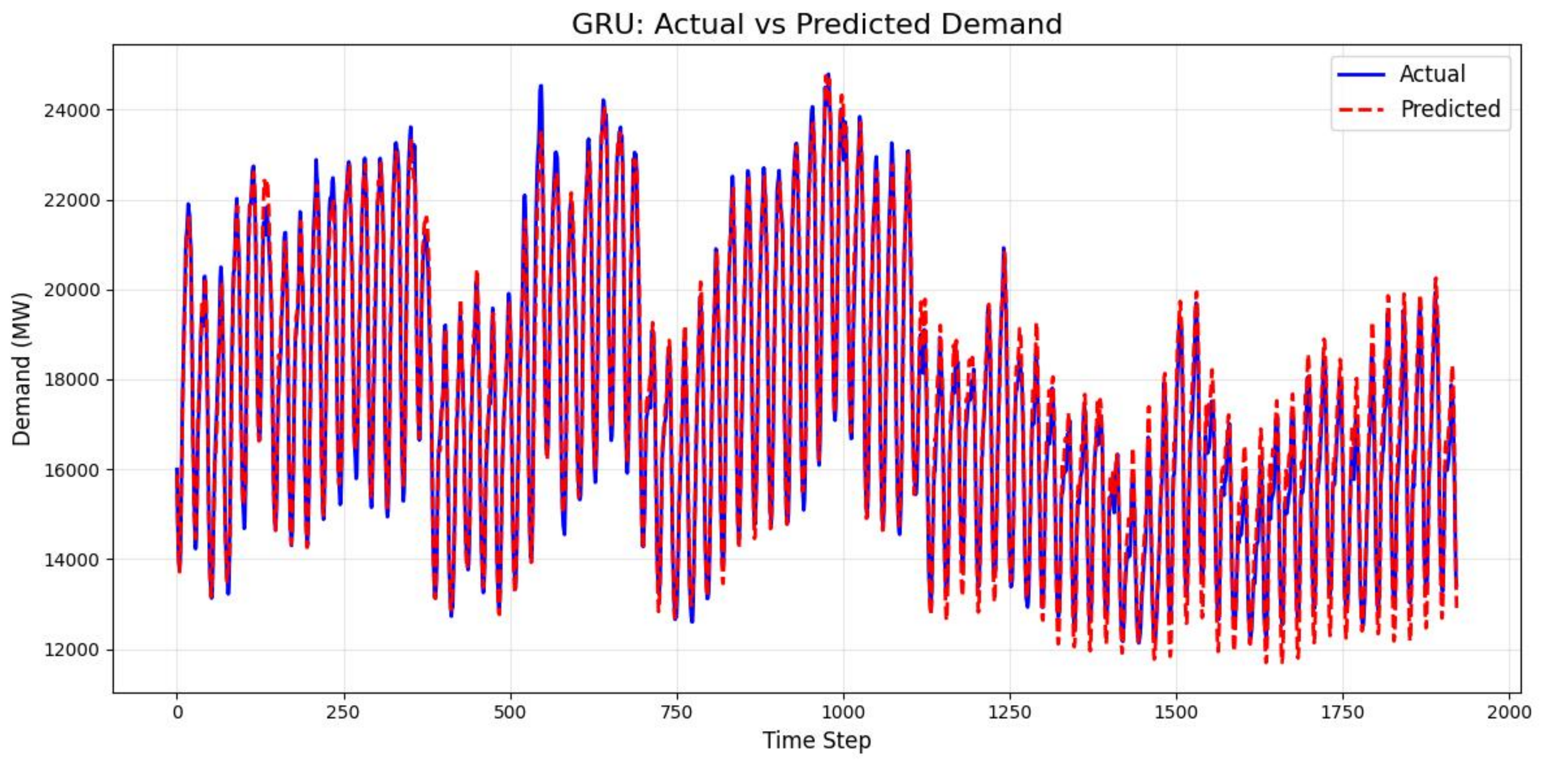
Since the dataset is looking at the Short-Term Dataset and make it simple, we are only trying to predict one number: Electricity demand for the next hour

Parameters:

- 50 neurons
- 0.2 dropout rate
- Implementing a dropout (to prevent overfitting)
- Dense layer of 1

```
def create_tcn_model(input_shape, neurons=50, dropout=0.2):
    model = Sequential([
        Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=input_shape, padding='causal'),
        MaxPooling1D(pool_size=2),
        Conv1D(filters=64, kernel_size=3, activation='relu', padding='causal'),
        MaxPooling1D(pool_size=2),
        Flatten(),
        Dense(neurons, activation='relu'),
        Dropout(dropout),
        Dense(1)
    ])
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')
    return model
```

GRU


GRU: Actual vs Predicted Demand
Actual
Predicted
Demand (MW)
24000
22000
20000
18000
16000
14000
12000
0
250
500
750
1000
1250
1500
1750
2000
Time Step

TCN

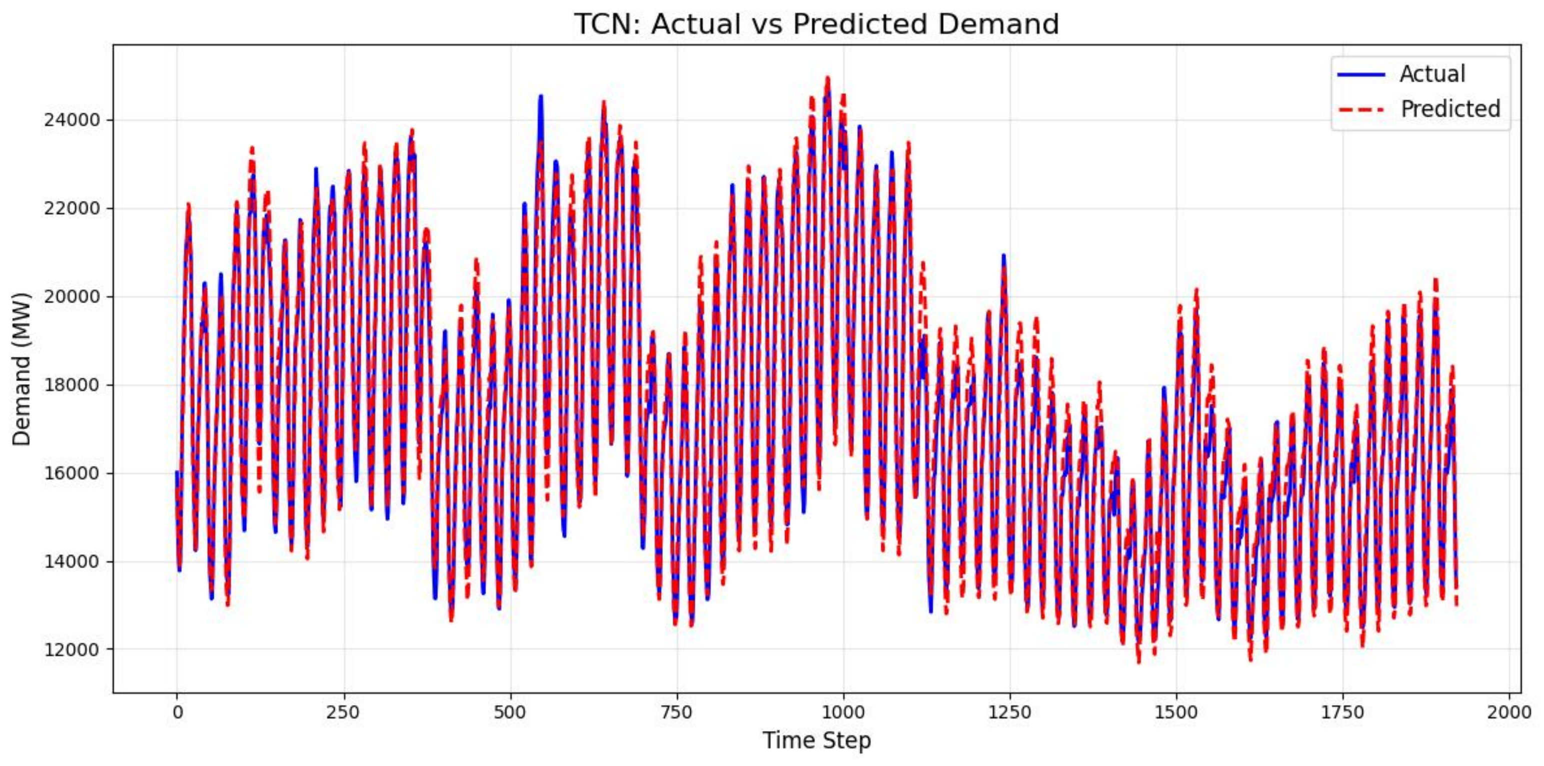
Since the dataset is looking at the Short-Term Dataset and predicting the consumption we want our model train on parameters on simple basic features and assess them on the output

Parameters:

- 50 neurons
- 0.2 dropout rate
- Filter Size of 64
- Kernel size of 3,
- Pool size of 2

```python
def create_tcn_model(input_shape, neurons=50, dropout=0.2):
    model = Sequential([
        Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=input_shape, padding='causal'),
        MaxPooling1D(pool_size=2),
        Conv1D(filters=64, kernel_size=3, activation='relu', padding='causal'),
        MaxPooling1D(pool_size=2),
        Flatten(),
        Dense(neurons, activation='relu'),
        Dropout(dropout),
        Dense(1)
    ])
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')
    return model
```

TCN



TCN: Actual vs Predicted Demand
Actual
Predicted
Demand (MW)
24000
22000
20000
18000
16000
14000
12000
0
250
500
750
1000
1250
1500
1750
2000
Time Step

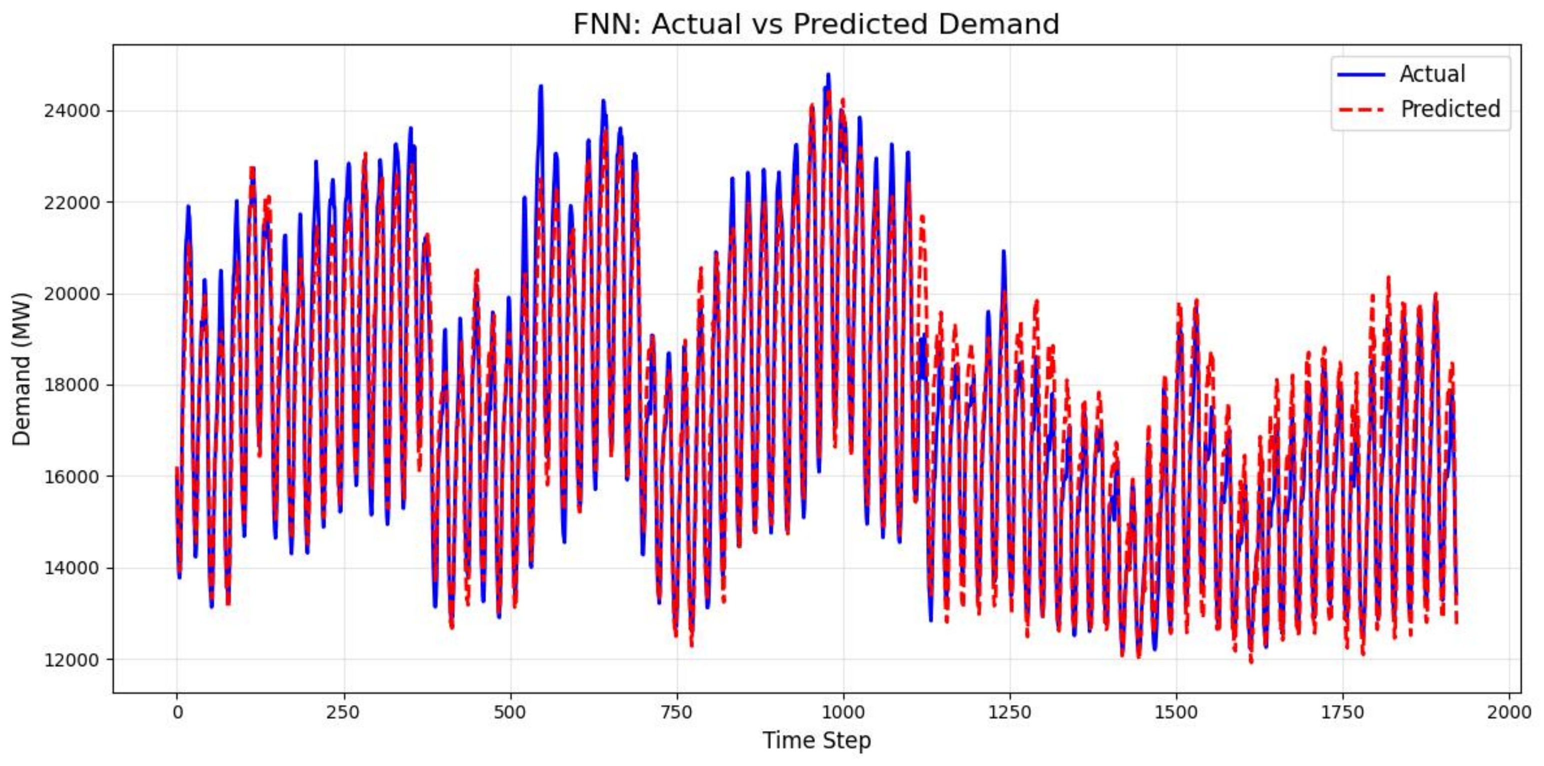
FNN

Parameters:

- 64 neurons
- 0.2 dropout rate
- Dense layer of 1
- RELU Activation

```
def create_fnn_model(input_shape, neurons=64, dropout=0.2):
    model = Sequential([
        Flatten(input_shape=input_shape),
        Dense(neurons, activation='relu'),
        Dropout(dropout),
        Dense(neurons, activation='relu'),
        Dropout(dropout),
        Dense(1)
    ])
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')
    return model
```

FNN



FNN: Actual vs Predicted Demand
Actual
Predicted
Demand (MW)
24000
22000
20000
18000
16000
14000
12000
0
250
500
750
1000
1250
1500
1750
2000
Time Step

Bidirectional LSTM

We want to look into different approach to train the data using both past and future values to train and predict Electricity demand for the next hour

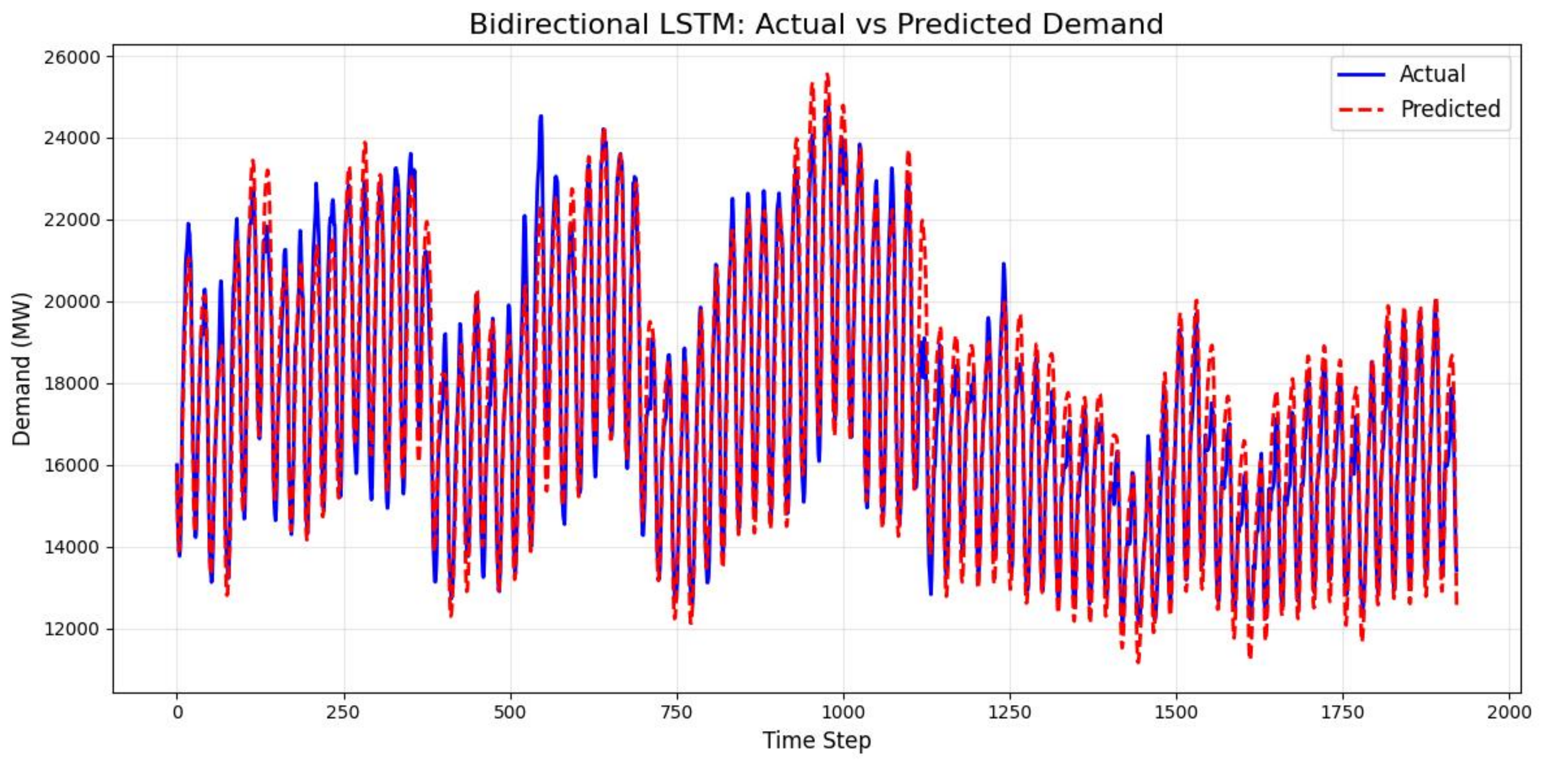
Parameters: (kept the same to see performance)

- 50 neurons
- 0.2 dropout rate
- Implementing a dropout (to prevent overfitting)
- RELU activation
- Dense layer of 1

We assumed that this model would be best because of the bi-directional reading

```python
def create_bidirectional_lstm_model(input_shape, neurons=50, dropout=0.2):
    model = Sequential([
        Bidirectional(LSTM(neurons, activation='relu', input_shape=input_shape, return_sequences=True)),
        Dropout(dropout),
        Bidirectional(LSTM(neurons, activation='relu')),
        Dropout(dropout),
        Dense(1)
    ])
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')
    return model
```

Bidirectional LSTM



Model Comparison