nn_model

January 8, 2024

1 Neural Network & Timeseries Forecasts

1.1 Using the Tensorflow Saved TCN

Load the Architecture and Weights Load the model like this:

```
from tensorflow.keras.models import load_model

MODEL_PATH = "./models/tcn"
tcn=load_model(MODEL_PATH)
```

Model needs the following features: - '**Max_Demand_GW**' - its autoregressive. - '**Plant_Production_GWh -'emissions_c02_GG**' - '**tavg**', - '**GDP_bln**' - '**date**' in `YYYY-DD-MM`,

Predict label is nex horizon's: '**Max_Demand_GW**'

Prepare Data with Utility The model is a simple one-step and needs a window of data (12 months), and will predict a horizon (next 1 month. or the 13th month). You need to prepare the data like so:

```
def prepare_data_and_windows(data_df, window=12, horizon=1):
    """
    Utility function to prepare the data. Assuming features are:
        - 'Plant_Production_GWh
        - 'emissions_c02_GG'
        - 'tavg',
        - 'GDP_bln'
        - 'date' in YYYY-DD-MM,
    :data_df dataframe: dataframe with `window_size` months of data to predict the `window_siz
    :param window_size: int, length of the input sequence
    :param horizon: int, forecasting horizon, defaults to 1
    :return: Array in the shape of (n_samples, n_steps, n_features)
    """
    MONTH_SINE = "month_sin"
    MONTH_COS = "month_cos"
    TARGET = "Max_Demand_GW"
    FEATURES = [
        "Plant_Production_GWh",
        "emissions_c02_GG",
```

```python
    "tavg",
    "GDP_bln",
]
INDEX = "Date"

def _encode_timewindows(data_df, features, target, window_size, horizon):
    """
    Create input and target windows suitable for TCN model.

    :param data: DataFrame with shape (n_samples, n_features)
    :param features: List of strings, names of the feature columns
    :param target: String, name of the target column
    :param window_size: int, length of the input sequence
    :param horizon: int, forecasting horizon
    :return: Array in the shape of (n_samples, n_steps, n_features)
    """
    X, y = [], []
    for i in tqdm(
        range(len(data_df) - window_size - horizon + 1), desc="Encoding Widows"
    ):
        input_window = data_df[features].iloc[i : i + window_size].values
        X.append(input_window)

        # Target window
        if horizon == 1:
            target_value = data_df[target].iloc[i + window_size]
        else:
            target_value = (
                data_df[target].iloc[i + window_size : i + window_size + horizon].values
            )
        y.append(target_value)
    return np.array(X), np.array(y)


def _encode_cyclics(data_df):
    """
    Encodes time cyclic features for a dataset with monthly sampling.
    Assuming we can capture the yearly periodicity by encoding the month as a wave.
    See: https://www.tensorflow.org/tutorials/structured_data/time_series#time
    :param data_df: The timeseries with a date in the format YYYY-DD-mm as index.
    :return: data_df with 2 new wave features.
    """
    months = data_df.index.month

    data_df[MONTH_SINE] = np.sin(2 * np.pi * months / 12)
    data_df[MONTH_COS] = np.cos(2 * np.pi * months / 12)

    return data_df
```

```python
    WINDOW_SIZE_MONTHS = 12
    EXT_FEATURES = FEATURES + [MONTH_SINE, MONTH_COS]

    normalizer = tf.keras.layers.experimental.preprocessing.Normalization(axis=-1)
    normalizer.adapt(data_df[FEATURES])
    data_df_normalized = normalizer(data_df[FEATURES])
    data_df_normalized = pd.DataFrame(
        data_df_normalized, columns=FEATURES, index=all_data_df.index
    )
    data_df_normalized = encode_cyclics(data_df_normalized)

    X, y = _encode_timewindows(
        pd.concat([data_df[TARGET], data_df_normalized], axis=1),
        EXT_FEATURES,
        TARGET,
        WINDOW_SIZE_MONTHS,
        PREDICTION_HORIZON,
    )
    return X, y, normalizer
```

Test Prediction Do a test prediction, example with our test dataset if we want to predict the max load for the last month:

```python
# Our test set has 2yrs, we get the last nov to nov window, and predict dec.
# if this slicing is confusing, we grab the last 13 months (+2) and slice it to before the 13t
window_12month_df = test_df.iloc[-(WINDOW_SIZE_MONTHS + 2) : -1]
ext_test_x, _, _ = prepare_data_and_windows(
    window_12month_df, window=WINDOW_SIZE_MONTHS, horizon=1
)
y_13th_month = val_model.predict(ext_test_x)
```

1.2 Experiment Details

In this notebook, we will build a Temporal Convolusion Network (TCN), which will learn and predict the electirity load demand from 20 years of timeseries data.

Convolutional neural networks (CNNs) are commonly used for time series forecasting tasks. The model takes the lagged time series as a 1D input signal, applies convolution and pooling operations to extract temporal features, and passes through fully connected layers to make prediction

Architecture inspired by: 1. Temporal Convolutional Networks Applied to Energy-Related Time Series Forecasting 2. Short-Term Load Forecasting Using Channel and Temporal Attention Based Temporal Convolutional Network 3. Zhang, Mingda. "Time series: Autoregressive models ar, ma, arma, arima." University of Pittsburgh (2018). 4. Augmented Dickey–Fuller test 5. Tensorflow Time series forecasting

1.2.1 Abbreviations

- CNN Convolutional Neural Network
- DL Deep Learning

- LSTM Long Short-Term Memory Network
- MAE Mean Absolute Error
- MIMO Multi-Input Multi-Output
- RNN Recurrent Neural Network
- TCN Temporal Convolutional Network
- WAPE Weighted Absolute Percentage Error

1.3 Outcome

The prediction target is: - A **Single-output**: **Max_Demand_GW** - A **Single-time-step**: **Next Month**

1.4 Evaluation Metric

Symmetric Mean Absolute Percentage Error (sMAPE) is the recommend metric to compare all our models:

$$\text{sMAPE}(y, o) = \frac{1}{N} \sum_{t=1}^{N} \frac{|y_t - o_t|}{(|y_t| + |o_t|)/2} \times 100$$

The paper recommends weighted absolute percentage error (WAPE):

$$\text{WAPE}(y, o) = \frac{\text{mean}(|y - o|)}{\text{mean}(y)}$$

where (y) and (o) are two vectors with the real and predicted values, respectively, that have a length equal to the forecasting horizon.

In energy forecasting, where load values can vary significantly, a weighted approach (like WAPE) helps prioritize the accuracy of predictions for higher-load periods. See Measuring forecast accuracy.

1.5 Dataset

Load, analyze, feature engineer and feature creation on: - 20 Years data, 2003-2022 - Data is cyclic, with accentuated 2012, 2015 & 2019 slope changes linked to the country's economic, social and political pivots. - Data is a timeseries, with monthly intervals. - Data has a unit root, with a rising trend due to population, industry and climate increase. - Data has a seasonality with low demand at spring and autumn, and highest demand end of summer.

Using the selected features in data_analysis.ipynb and the feature importance by pca_analysis.ipynb.

[1]:
```python
from datetime import datetime
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tqdm import tqdm

VAL_SPLIT = 0.2
DATA_PATH = "./data"
END_DATE = datetime(2022, 12, 31)
```

```python
all_data_df = pd.read_csv(f"{DATA_PATH}/all_data.csv", index_col=0,␣
 ↪parse_dates=True)
train_df = pd.read_csv(f"{DATA_PATH}/train_data.csv", index_col=0,␣
 ↪parse_dates=True)
test_df = pd.read_csv(f"{DATA_PATH}/test_data.csv", index_col=0,␣
 ↪parse_dates=True)
test_df = test_df[test_df.index <= END_DATE]

print(f"Shapes: train_df: {train_df.shape} test_df: {test_df.shape}")

TARGET = "Max_Demand_GW"
FEATURES = [
    "Plant_Production_GWh",
    "emissions_c02_GG",
    "tavg",
    "GDP_bln",
]
INDEX = "Date"

test_df[FEATURES].head(3)
```

```
Shapes: train_df: (192, 10) test_df: (48, 10)
```

[1]:

	Plant_Production_GWh	emissions_c02_GG	tavg	GDP_bln
Date				
2019-01-01	224.76	75.11	11.6	14.19
2019-02-01	199.54	73.41	12.0	14.19
2019-03-01	199.28	66.58	14.5	14.19

1.6 Baseline Model

Let's start with a timeseries definition.

A timeseries is comprised of 4 components, in a multiplicative timeseries (where the components are interdependant) it is visualized as follows:

$Y(t) = T(t) \times S(t) \times C(t) \times \epsilon(t)$

Where T is the trend, S is the seasonal trend, C is the cyclical trend and ϵ is noise in any point in time t. The additive version (no dependant components) would substitute the operator to $+$.

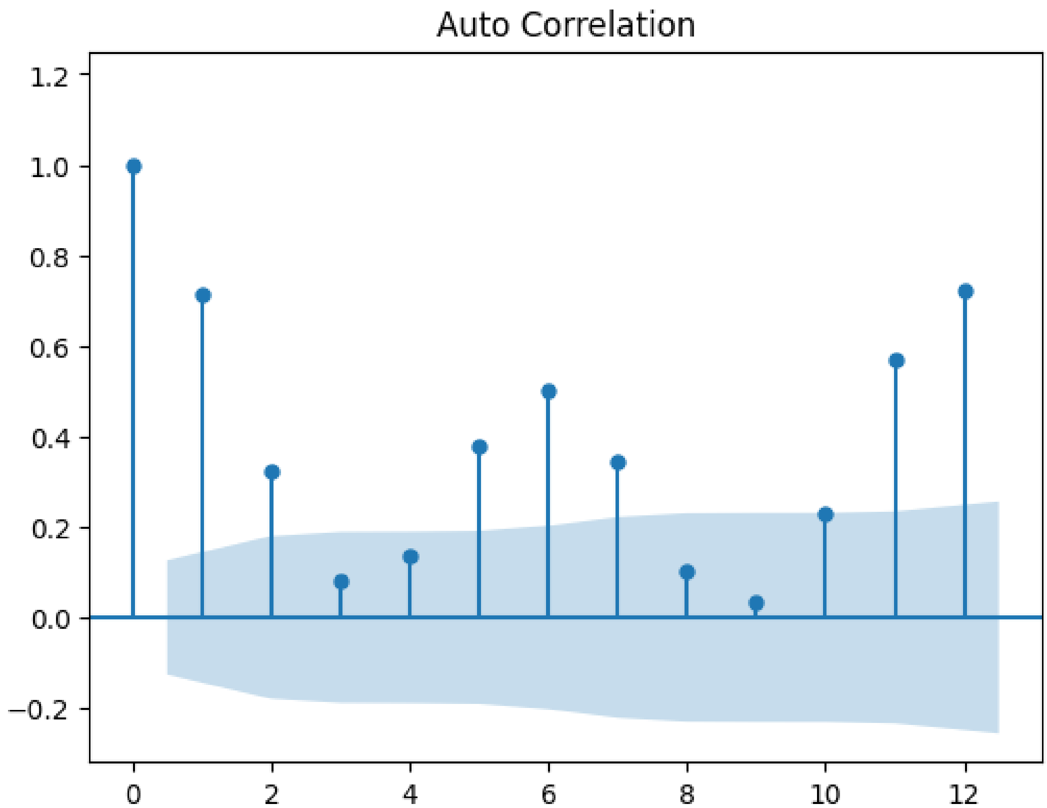
We know its a seasonal and probably not stationary timeseries - we verify these assumptions below using statsmodels's seasonal_decompose api:

[2]:

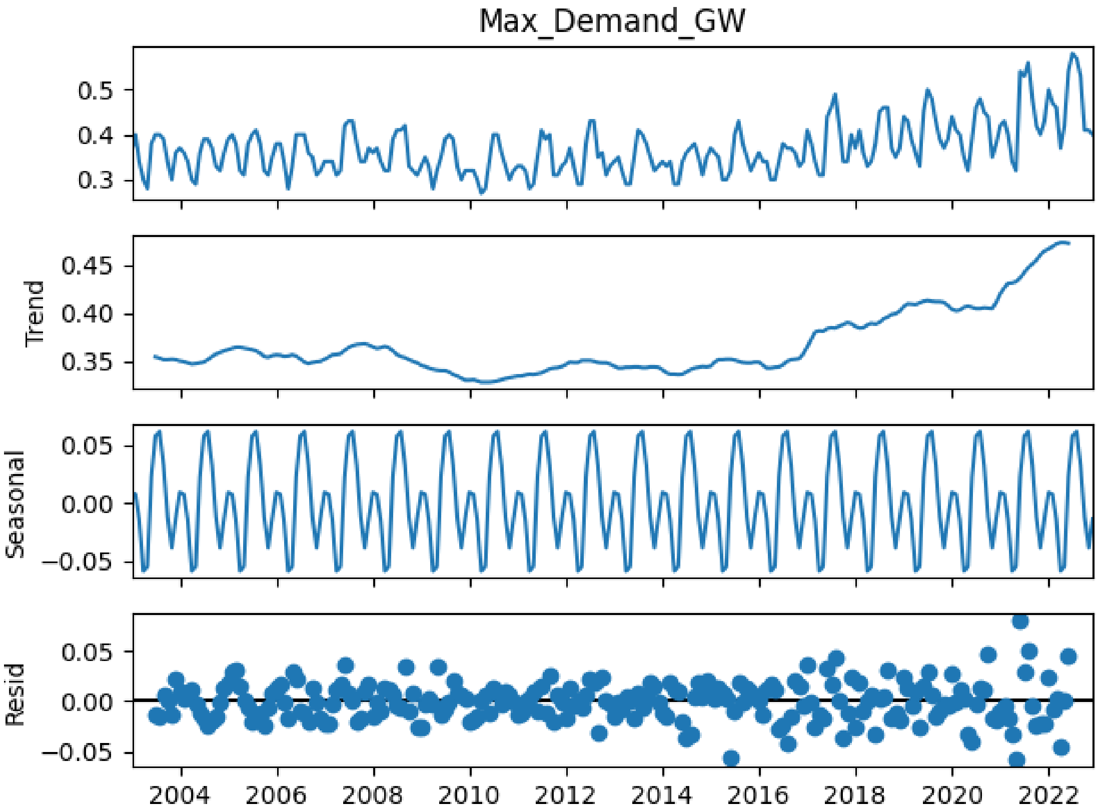
```python
import statsmodels.api as sm
from statsmodels.tsa.seasonal import seasonal_decompose


def wmape(y, ypred):
```

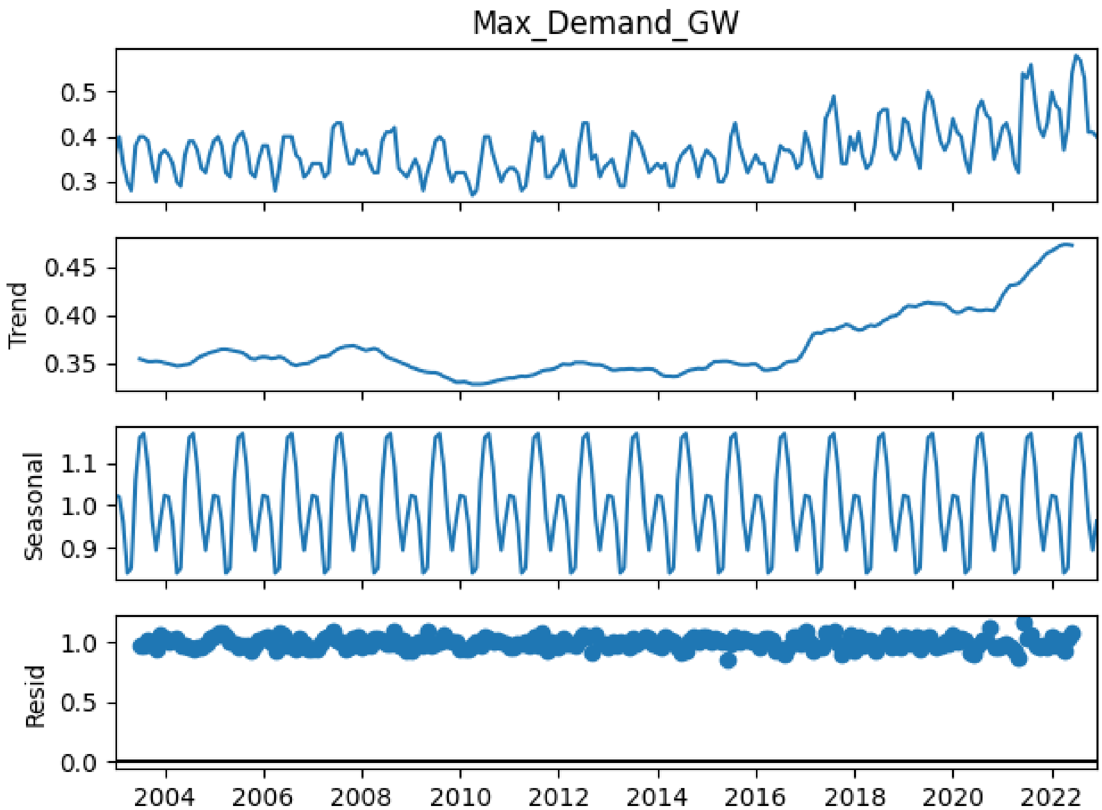
```
    """
    Calculate Weighted Mean Absolute Percentage Error (WMAPE).
    Custom error score for these NNs.

    Parameters:
    - y: numpy array, actual values
    - ypred: numpy array, predicted values

    Returns:
    - wape: float, WAPE value
    """
    absolute_errors = np.abs(y - ypred)
    wape = np.mean(absolute_errors) / np.mean(y) * 100
    return wape


PREDICTION_HORIZON = 1
LAGS = 1
all_data_ts = all_data_df[TARGET]

plt.figure(figsize=(12, 6))
sm.graphics.tsa.plot_acf(
    all_data_ts,
    lags=LAGS * 12,
    auto_ylims=True,
    title="Auto Correlation",
)

a_season = seasonal_decompose(all_data_ts, model="a", period=12)
m_season = seasonal_decompose(all_data_ts, model="m", period=12)
fig = a_season.plot()
fig = m_season.plot()
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



Auto Correlation
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
2
4
6
8
10
12


Max_Demand_GW
0.5
0.4
0.3
Trend
0.45
0.40
0.35
Seasonal
0.05
0.00
−0.05
Resid
0.05
0.00
−0.05
2004
2006
2008
2010
2012
2014
2016
2018
2020
2022



First plot is an **Auto Correlation Function (ACF)**:

The ACF function measures the linear predictability of Yt using adjacent points in 2 timeseries (s and t), or the lagged versions of a time series. It provides a pearson value between -1 and 1. The ACF is defined as:

$$\rho(s,t) = \frac{\gamma(s,t)}{\sqrt{\gamma(s,s)\gamma(t,t)}}$$

Where: - \$ (s, t) \$ is the autocorrelation function between two time points \$ s\$ and t. - $\gamma(s,t)$ is the autocovariance function between the same two time points. - The denominator $\sqrt{\gamma(s,s)\gamma(t,t)}$ is the product of the square roots of the autocovariances at times s and t, ensuring that the autocorrelation is scaled between -1 and 1.

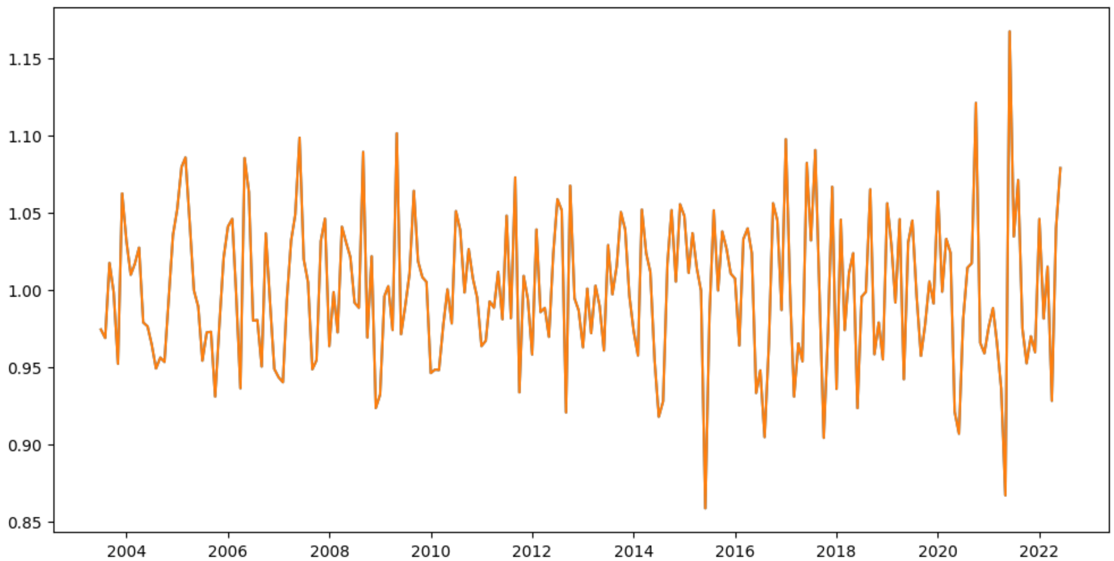
There is significant autoregression at lags *1,2,11,12* between a pearson of *0.6* to *0.8* - indicating this dataset is Auto Regressive (AR). The shaded area represents values outside the 95% confidence interval.

Second plots are **Seasonal Decompositions** (T, S and C weights through time) for additive and multiplicative data: - **Additive Model**: Used when there are weak variations around a trend. (seasonality is consistent in magnitude over time) - in the case of this dataset, it only created increasing noise. - **Multiplicative Model**: Assumes variations are proportional to the level of the time series (seasonality increases in magnitude as the series increases). This suggests that there is a **unit root**.

Unit root is a characteristic of a time series that makes it non-stationary. A stationary timeseries has a finite variance and a constant mean. Therefore a model would remove this unit root differencing the data: - $ Y_t = Y_t - Y_{t-1} $ - $ ^2 Y_t = Y_t - Y_{t-1} = (Y_t - Y_{t-1}) - (Y_{t-1} - Y_{t-2}) $ - $ ^d Y_t = (^{d-1} Y_t) $

Where Y is a point in time and $ ^d Y_t $ is used when $ (d > 2) $

[3]:
```
flat_target = all_data_ts.values / m_season.seasonal / m_season.trend

plt.figure(figsize=(12, 6))
plt.plot(all_data_ts.index, flat_target, flat_target)
plt.show()
```



We store the trend wieghts for predictions with non-timeseries based models.

[4]:
```
df_adjustment = pd.DataFrame(
    {
        "Seasonal": m_season.seasonal,
        "Trend": m_season.trend,
        "Month": all_data_ts.index.month,
    }
)

df_adjustment = df_adjustment.groupby("Month").mean()
df_adjustment.reset_index(inplace=True)

# Save weights for future prediction
```

```python
df_adjustment.to_pickle(f"{DATA_PATH}/seasonal_adjustment.pkl")
df_adjustment
```

[4]:

	Month	Seasonal	Trend
0	1	1.023660	0.367061
1	2	1.019938	0.367829
2	3	0.960148	0.368509
3	4	0.841466	0.368969
4	5	0.852615	0.369364
5	6	1.058592	0.369693
6	7	1.157900	0.363904
7	8	1.168566	0.364364
8	9	1.090147	0.364781
9	10	0.968335	0.365197
10	11	0.894940	0.365658
11	12	0.963693	0.366316

Finally we verify test the Null Hypothesis $H0$ that the data is not stationary, using an augmented Dickey–Fuller test (ADF) which solves the following equation:

$$\Delta Y_t = \alpha + \beta_t + \gamma Y_{t-1} + \delta_1 \Delta Y_{t-1} + \delta_2 \Delta Y_{t-2} + \cdots + \delta_p \Delta Y_{t-p} + \varepsilon_t$$

Where: - ΔY_t is the differenced series. - α is a constant term and β_t is a trend term. - γ is the coefficient of Y_{t-1}, the lagged value of the series, as is $\delta_1, \delta_2 ..., \delta_1 p$ of the lagged differenced terms with p. - ε_t is the error.

$\gamma = 0$, which implies the presence of a unit root its compliment $\gamma < 0$ indicates stationarity rejecting the $H0$.

Using the adfuller function, it returns a T statistic and P-value are positive and large, meaning $H0$ is not rejected and our data is not stationary and has a unit root. For a stationary timeseries: *P-value significance level*, and *T-stat critical value*.

[5]:
```python
from statsmodels.tsa.stattools import adfuller

result = adfuller(all_data_ts.values)
print("T-stat: %f" % result[0])
print("p-value: %f" % result[1])
for key, value in result[4].items():
    print("Critial Values:")
    print(f"   {key}, {value}")
```

```
T-stat: 0.212656
p-value: 0.972963
Critial Values:
   1%, -3.459884913337196
Critial Values:
   5%, -2.8745310704320794
Critial Values:
   10%, -2.573693840082908
```

1.7 Auto ARIMA model

ARIMA is an AutoRegressive Integrated Moving Average model.

An Autoregressive timeseries model that predicts $Y - t$ can be approximated as follows:

$ Y_t = t + _{i=1}^{t} _i $

Where δt is the drift by its components, and $ _i $ is noise throughout time t.

Constructing the ARIMA baseline model is beyond this assignment. Instead, we will use the AutoArima package to create a good enough model - we will do no further finetuning or analysis, but we use the baseline modl to compare the TCN performance.

ARIMA models are generally denoted `ARIMA(p,d,q)(P, D, Q)`, where - `p` is the number of time lags, - `d` is the degree of differencing, - `q` is the order of the moving-average model, - `P, D, Q` is the above for the seasonal component.

1.8 Baseline Experiment

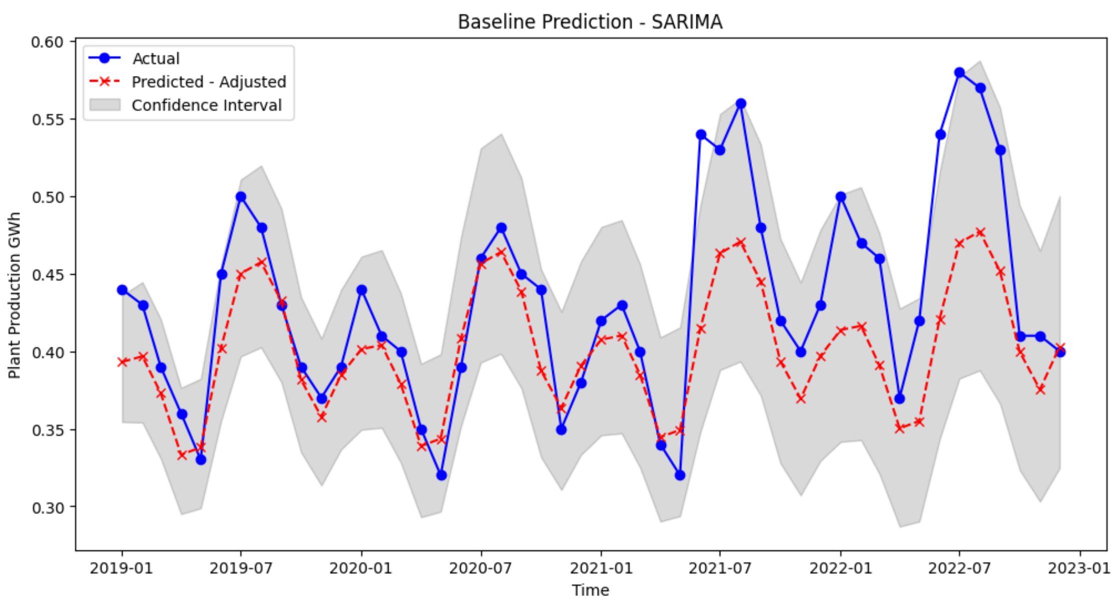
The code below builds, fits and predicts the baseline AutoARIMA model:

[6]:
```
from sklearn.metrics import mean_squared_error
from pmdarima.metrics import smape
from pmdarima.pipeline import Pipeline
from pmdarima.preprocessing import BoxCoxEndogTransformer
from pmdarima.arima import auto_arima


def train_fit_base_arima():
    """
    Utility to create the pipelines for the baseline model.
    See: https://alkaline-ml.com/pmdarima/modules/generated/pmdarima.arima.
 ↪auto_arima.html?highlight=auto_arima
    """
    train_df.index.names = [INDEX]
    arima = Pipeline(
        [
            ("boxcox", BoxCoxEndogTransformer(lmbda=0)),
            (
                "arima",
                auto_arima(
                    y=train_df[TARGET],
                    d=PREDICTION_HORIZON,  # Diffs
                    stationary=False,  # Has a trend
                    start_p=LAGS,  # Lag
                    m=12,  # Monthly seasonality
                    seasonal=True,
                    maxiter=30,
                    with_intercept=True,
                    information_criterion="bic",
```

```python
                    scoring="mse",
                    stepwise=True,
                    error_action="ignore",
                    trace=False,  # Set to TRUE to see training.
                ),
            ),
        ]
    )

    arima.fit(train_df[TARGET])
    return arima


def run_base_experiment():
    """
    Train it on the train dataset.
    Estimate steps up to the lenght of the test dataset.

    No exog data will be used, this is a baseline model (and for simplicity,␣
 ↪else we have to do recursive stepwise predictions)
    """
    arima = train_fit_base_arima()
    fc, co_int = arima.predict(
        n_periods=len(test_df), return_conf_int=True, inverse_transform=True
    )  # 4 Years

    print(f"RMSE: {mean_squared_error(test_df[TARGET], fc, squared=False):0.
 ↪02f}")
    print(f"SMAPE: {smape(test_df[TARGET], fc):0.02f}%")
    print(f"WMAPE: {wmape(test_df[TARGET], fc):0.02f}%")

    plt.figure(figsize=(12, 6))
    plt.plot(test_df[TARGET], label="Actual", color="blue", marker="o")
    plt.plot(
        test_df[TARGET].index,
        fc,
        label="Predicted - Adjusted",
        color="red",
        linestyle="dashed",
        marker="x",
    )
    plt.fill_between(
        test_df.index,
        co_int[:, 0],
        co_int[:, 1],
        color="k",
        alpha=0.15,
```

```
        label="Confidence Interval",
    )

    plt.xlabel("Time")
    plt.ylabel("Plant Production GWh")
    plt.title("Baseline Prediction - SARIMA")
    plt.legend()
    plt.show()


run_base_experiment()
```

```
RMSE: 0.05
SMAPE: 8.22%
WMAPE: 8.36%
```

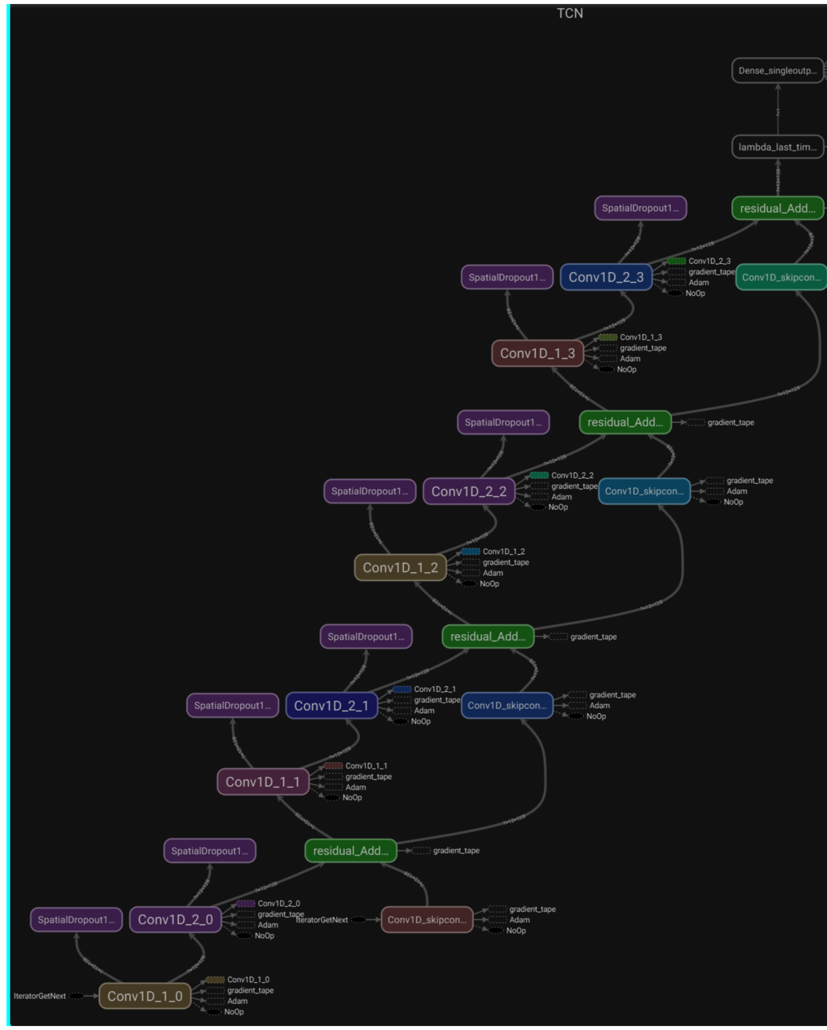


`Best model:  ARIMA(1,1,1)(1,0,1)[12]` means autoarima identified 1 lag across all its components except the degree of difference for the seasonal, which is stationary.

2 Temporal Convolution Neural Network (TCN)

CNNs, RNNs and other stateful deeplearning models are used for timeseries (including transformers, as language is also a timeseries). The TCN is made specifically for timeseries problems, following the paper's architecture - we also migrate their code to tensorflow2 from a legacy version of keras used in their paper: 1. Input 3D Tensor of shape (batch_size, window_size, n_features) 2. Output 2D tensor of shape (batch_size, horizon) 3. 6 hidding layers (from the paper's code) comprised of: 1. x2 1D convulations with relu activation and a spatial dropouts. 2. 1D Dilated Convolution to capture residuals with linear activation. 3. An addition layer to add back the residuals into the next layers input 4. a single dense layer fto output the next timestep according to the given horizon. 5. ADAM learning optimizer configured according to the paper. 6. Fast stop function configured according to the paper.

The architecture is visualized in the graph below:




```
[17]: from tensorflow.keras.layers import (
    SpatialDropout1D,
    Dense,
    Conv1D,
    Layer,
    Normalization,
    Add,
    Input,
    Lambda,
)
from tensorflow.keras import Model


class TCNBlock(Layer):
```

```python
    """
    TCN Residual Block that uses zero-padding to maintain `steps` value of the␣
↪ouput equal to the one in the input.
    Residual Block is obtained by stacking togeather (2x) the following:
        - 1D Dilated Convolution
        - ReLu
        - Spatial Dropout
    And adding the input after trasnforming it with a 1x1 Conv
    forked and extended from: https://github.com/albertogaspar/dts/blob/master/
↪dts/models/TCN.py
    """

    def __init__(
        self,
        filters=1,
        kernel_size=2,
        dilation_rate=None,
        kernel_initializer="glorot_normal",
        bias_initializer="glorot_normal",
        kernel_regularizer=None,
        bias_regularizer=None,
        use_bias=False,
        dropout_rate=0.0,
        id=None,
        **kwargs,
    ):
        """ "
        Arguments
            filters: Integer, the dimensionality of the output space
                (i.e. the number of output filters in the convolution).
            kernel_size: An integer or tuple/list of a single integer,
                specifying the length of the 1D convolution window.
            dilation_rate: an integer or tuple/list of a single integer,␣
↪specifying
                the dilation rate to use for dilated convolution.
                Usually dilation rate increases exponentially with the depth of␣
↪the network.
            activation: Activation function to use
                If you don't specify anything, no activation is applied
                (ie. "linear" activation: `a(x) = x`).
            use_bias: Boolean, whether the layer uses a bias vector.
            kernel_initializer: Initializer for the `kernel` weights matrix
            bias_initializer: Initializer for the bias vector
            kernel_regularizer: Regularizer function applied to the `kernel`␣
↪weights matrix
            bias_regularizer: Regularizer function applied to the bias vector
                (see [regularizer](../regularizers.md)).
```

```
        # Input shape
            3D tensor with shape: `(batch, steps, n_features)`
        # Output shape
            3D tensor with shape: `(batch, steps, filters)`
        """
        super(TCNBlock, self).__init__(**kwargs)
        self.filters = filters
        self.kernel_size = kernel_size
        self.dilation_rate = dilation_rate

        # Capture feature set from the input
        self.conv1 = Conv1D(
            filters=filters,
            kernel_size=kernel_size,
            use_bias=use_bias,
            bias_initializer=bias_initializer,
            bias_regularizer=bias_regularizer,
            kernel_initializer=kernel_initializer,
            kernel_regularizer=kernel_regularizer,
            padding="causal",
            dilation_rate=dilation_rate,
            activation="relu",
            name=f"Conv1D_1_{id}",
        )

        # Spatial dropout is specific to convolutions by dropping an entire␣
↪timewindow,
        # not to rely too heavily on specific features detected by the kernels.
        self.dropout1 = SpatialDropout1D(
            dropout_rate, trainable=True, name=f"SpatialDropout1D_1_{id}"
        )
        # Capture a higher order feature set from the previous convolution
        self.conv2 = Conv1D(
            filters=filters,
            kernel_size=kernel_size,
            use_bias=use_bias,
            bias_initializer=bias_initializer,
            bias_regularizer=bias_regularizer,
            kernel_initializer=kernel_initializer,
            kernel_regularizer=kernel_regularizer,
            padding="causal",
            dilation_rate=dilation_rate,
            activation="relu",
            name=f"Conv1D_2_{id}",
        )
        self.dropout2 = SpatialDropout1D(
            dropout_rate, trainable=True, name=f"SpatialDropout1D_2_{id}"
```

```
        )

        # The skip connection is an addition of the input to the block with the␣
↪output of the second dropout layer.
        # Solves vanishing gradient, carries info from earlier layers to later␣
↪layers, allowing gradients to flow across this alternative path.
        # Does not learn direct mappings, but differences (residuals) while␣
↪keeping temporal context.
        # Note how it keeps dims intact with kernel 1.
        self.skip_out = Conv1D(
            filters=filters,
            kernel_size=1,
            activation="linear",
            name=f"Conv1D_skipconnection_{id}",
        )
        # This is the elementwise add for the residual connection and Conv1d␣
↪2's output
        self.residual_out = Add(name=f"residual_Add_{id}")

    def apply_block(self, inputs):
        x = self.conv1(inputs)
        x = self.dropout1(x)
        x = self.conv2(x)
        x = self.dropout2(x)

        # Residual output by adding the inputs back:
        skip_out_x = self.skip_out(inputs)
        x = self.residual_out([x, skip_out_x])
        return x


def TCN(
    input_shape,
    output_horizon=1,
    num_filters=32,
    num_layers=1,
    kernel_size=2,
    dilation_rate=2,
    kernel_initializer="glorot_normal",
    bias_initializer="glorot_normal",
    kernel_regularizer=None,
    bias_regularizer=None,
    use_bias=False,
    dropout_rate=0.0,
):
    """
    Tensorflow TCN Model builder.
```

```
    forked and extended from: https://github.com/albertogaspar/dts/blob/master/
↪dts/models/TCN.py
    see: https://www.tensorflow.org/api_docs/python/tf/keras/Model
    see: https://www.tensorflow.org/guide/keras/
↪making_new_layers_and_models_via_subclassing#the_model_class
    see: https://www.tensorflow.org/api_docs/python/tf/keras/regularizers/L2

    :param layers: int
        Number of layers for the network. Defaults to 1 layer.
    :param filters: int
        the number of output filters in the convolution. Defaults to 32.
    :param kernel_size: int or tuple
        the length of the 1D convolution window
    :param dilation_rate: int
        the dilation rate to use for dilated convolution. Defaults to 1.
    :param output_horizon: int
        the output horizon.
    """
    x = inputs = Input(shape=input_shape)
    for i in range(num_layers):
        block = TCNBlock(
            filters=num_filters,
            kernel_size=kernel_size,
            dilation_rate=dilation_rate**i,
            kernel_initializer=kernel_initializer,
            bias_initializer=bias_initializer,
            kernel_regularizer=kernel_regularizer,
            bias_regularizer=bias_regularizer,
            use_bias=use_bias,
            dropout_rate=dropout_rate,
            id=i,
        )
        x = block.apply_block(x)
    # Selects the last timestep and predict the demand in the 1 DIM layer.
    x = Lambda(lambda x: x[:, -1:, 0], name="lambda_last_timestep")(x)
    outputs = Dense(output_horizon, name="Dense_singleoutput")(x)

    model = Model(inputs=inputs, outputs=outputs, name="TCN")
    return model
```

2.1 Encoding Time Windows, Data Prep and Normalization

Encode the timesteps and windows for a recurrent network. We also normalize and check the raw values.

Tensorflow recommends encoding date as a sine & cosine wave.

[18]:
```python
def encode_timewindows(data_df, features, target, window_size, horizon):
    """
    Create input and target windows suitable for TCN model.

    :param data: DataFrame with shape (n_samples, n_features)
    :param features: List of strings, names of the feature columns
    :param target: String, name of the target column
    :param window_size: int, length of the input sequence.
    :param horizon: int, forecasting horizon.
    :return: Array in the shape of (n_samples, n_steps, n_features)
    """
    X, y = [], []
    for i in tqdm(
        range(len(data_df) - window_size - horizon + 1), desc="Encoding Widows"
    ):
        input_window = data_df[features].iloc[i : i + window_size].values
        X.append(input_window)

        # Target window, note it predicts {horizon} steps ahead
        if horizon == 1:
            target_value = data_df[target].iloc[i + window_size]
        else:
            target_value = (
                data_df[target].iloc[i + window_size : i + window_size +␣
↪horizon].values
            )
        y.append(target_value)
    return np.array(X), np.array(y)


MONTH_SINE = "month_sin"
MONTH_COS = "month_cos"


def encode_cyclics(data_df):
    """
    Encodes time cyclic features for a dataset with monthly sampling.
    Assuming we can capture the yearly periodicity by encoding the month as a␣
↪wave.
    See: https://www.tensorflow.org/tutorials/structured_data/time_series#time
    :param data_df: The timeseries with a date in the format YYYY-DD-mm as␣
↪index.
    :return: data_df with 2 new wave features.
    """
    months = data_df.index.month

    data_df[MONTH_SINE] = np.sin(2 * np.pi * months / 12)
```

```python
    data_df[MONTH_COS] = np.cos(2 * np.pi * months / 12)
    return data_df


WINDOW_SIZE_MONTHS = 12
EXT_FEATURES = FEATURES + [TARGET, MONTH_SINE, MONTH_COS]


def prepare_data_and_windows(
    data_df, window=WINDOW_SIZE_MONTHS, horizon=PREDICTION_HORIZON
):
    """
    Utility function to prepare the data. Assuming features are:
        - 'Plant_Production_GWh
        - 'emissions_cO2_GG'
        - 'tavg',
        - 'GDP_bln'
        - 'date' in YYYY-DD-MM,
    :data_df dataframe: dataframe with `window_size` months of data to predict␣
 ↪the `window_size`+`horizon`.
    :param window_size: int, length of the input sequence
    :param horizon: int, forecasting horizon, defaults to 1
    :return: Array in the shape of (n_samples, n_steps, n_features)

    """
    normalizer = Normalization(axis=-1)

    normalizer.adapt(data_df[FEATURES])
    data_df_normalized = normalizer(data_df[FEATURES])
    data_df_normalized = pd.DataFrame(
        data_df_normalized, columns=FEATURES, index=data_df.index
    )
    data_df_normalized = encode_cyclics(data_df_normalized)
    X, y = encode_timewindows(
        pd.concat([data_df[TARGET], data_df_normalized], axis=1),
        EXT_FEATURES,
        TARGET,
        window,
        horizon,
    )
    print(
        f"FEATURES: {EXT_FEATURES}, TARGET: '{TARGET}', window:␣
 ↪{WINDOW_SIZE_MONTHS}, horizon: {PREDICTION_HORIZON}"
    )
    print(
        f"Shape unencoded (including target label and superflous features):␣
 ↪{data_df.shape}"
```

```
    )
    print(f"Shape encoded (window and selected exog features only): {X.shape}")

    return X, y, normalizer


# Yes, we are using the whole dataset not the training dataset.
# See: https://www.tensorflow.org/api_docs/python/tf/keras/Model#fit
# we will tell keras to do a validation split, it will not fit on the␣
 ↪validation data.

X, y, normalizer = prepare_data_and_windows(all_data_df)
print(f"Label shape encoded: {y.shape}")
print(f"First window exog normalized: {X[0, 1:, :]}")
print(f"First window targets: {y[0:11]}")


input_shape = (WINDOW_SIZE_MONTHS, X.shape[2])
input_shape
```

```
Encoding Widows: 100%|      | 228/228 [00:00<00:00, 1015.94it/s]

FEATURES: ['Plant_Production_GWh', 'emissions_c02_GG', 'tavg', 'GDP_bln',
'Max_Demand_GW', 'month_sin', 'month_cos'], TARGET: 'Max_Demand_GW', window: 12,
horizon: 1
Shape unencoded (including target label and superflous features): (240, 10)
Shape encoded (window and selected exog features only): (228, 12, 7)
Label shape encoded: (228,)
First window exog normalized: [[-2.43314236e-01 -6.02663793e-02 -1.68626821e+00
-1.15034938e+00
   4.00000000e-01  8.66025404e-01  5.00000000e-01]
 [-5.86296439e-01 -7.57323503e-01 -1.25902200e+00 -1.15034938e+00
   3.40000000e-01  1.00000000e+00  6.12323400e-17]
 [-1.22047710e+00 -1.22202861e+00 -7.38896132e-01 -1.15034938e+00
   3.00000000e-01  8.66025404e-01 -5.00000000e-01]
 [-9.82659221e-01 -1.45438099e+00  1.71324030e-01 -1.15034938e+00
   2.80000000e-01  5.00000000e-01 -8.66025404e-01]
 [-2.37165261e-02 -2.92618752e-01  1.23015177e+00 -1.15034938e+00
   3.80000000e-01  1.22464680e-16 -1.00000000e+00]
 [ 1.22226954e+00 -6.02663793e-02  1.73170149e+00 -1.15034938e+00
   4.00000000e-01 -5.00000000e-01 -8.66025404e-01]
 [ 1.03495598e+00 -6.02663793e-02  1.71312582e+00 -1.15034938e+00
   4.00000000e-01 -8.66025404e-01 -5.00000000e-01]
 [-1.89933151e-01 -1.76442564e-01  9.32936907e-01 -1.15034938e+00
   3.90000000e-01 -1.00000000e+00 -1.83697020e-16]
 [-3.72452140e-01 -7.57323503e-01  4.87114847e-01 -1.15034938e+00
   3.40000000e-01 -8.66025404e-01  5.00000000e-01]
 [-1.03252411e+00 -1.22202861e+00 -2.74498045e-01 -1.15034938e+00
```

```
  3.00000000e-01 -5.00000000e-01  8.66025404e-01]
 [-4.30627733e-01 -5.24971128e-01 -1.03611100e+00 -1.15034938e+00
  3.60000000e-01 -2.44929360e-16  1.00000000e+00]]
First window targets: [0.37 0.36 0.34 0.3  0.29 0.36 0.39 0.39 0.37 0.33 0.32]
```

[18]:

```
(12, 7)
```

2.2 Fit and Train TCN

Build and train the TCN according to the current parameters selected.

[27]:

```python
from tensorflow.keras.regularizers import L2
from tensorflow.keras.callbacks import EarlyStopping, TensorBoard
from tensorflow.keras.optimizers import Adam
from datetime import datetime
from sklearn.model_selection import ParameterGrid

EPOCHS = 300
BATCH_SIZE = 32
FILTER = 128
DROPRATE = 0.5
POOL_SIZE = 2
KERNEL_SIZE = 4
DILATION_RATE = 4
MAX_LAYERS = 4
L2_REG = 0.005
LEARN_RATE = 0.0001
MODEL_LOG_DIR = f'./logs/{datetime.now().strftime("%m%d-%H%M%S")}'
# See: https://scikit-learn.org/stable/modules/generated/sklearn.
↪model_selection.ParameterGrid.html
GRID = {
    "num_filters": [32, 64, 128],
    "kernel_size": [2, 3, 4],
    "dilation_rate": [1, 2, 4],
    "dropout_rate": [0.1, 0.2, 0.3],
    "num_layers": [6, 5, 3],
    "l2_reg": [0.005, 0.001, 0.01],
    "learning_rate": [0.001, 0.01, 0.1],
}

print(f"Model logs for Tensorboard available here: {MODEL_LOG_DIR}")


def build_tcn(X, y):
    model = TCN(
        input_shape=input_shape,
```

```
        output_horizon=PREDICTION_HORIZON,
        num_filters=FILTER,
        kernel_size=KERNEL_SIZE,
        num_layers=MAX_LAYERS,
        dilation_rate=DILATION_RATE,
        kernel_regularizer=L2(l2=L2_REG),
        bias_regularizer=L2(l2=L2_REG),
    )

    # See: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam
    optimizer = Adam(LEARN_RATE)
    metrics = ["mse", "mae", "mape"]
    model.compile(loss="mse", optimizer=optimizer, metrics=metrics)

    # Paper's `patience` was 50, we limited to 25 and watch the MAPE
    # see: https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/
↪EarlyStopping
    callbacks = [
        EarlyStopping(patience=25, monitor="val_mape",␣
↪restore_best_weights=True),
        TensorBoard(log_dir=MODEL_LOG_DIR),
    ]
    history = model.fit(
        X,
        y,
        validation_split=VAL_SPLIT,
        shuffle=False,
        epochs=EPOCHS,
        batch_size=BATCH_SIZE,
        callbacks=callbacks,
        verbose=1,
    )
    return model, history


model, history = build_tcn(X, y)
model.summary()
```

```
Model logs for Tensorboard available here: ./logs/0106-152816
Epoch 1/300
6/6 [==============================] - 4s 350ms/step - loss: 4.6373 - mse:
0.1285 - mae: 0.3551 - mape: 100.1481 - val_loss: 4.6483 - val_mse: 0.1979 -
val_mae: 0.4382 - val_mape: 101.0546
Epoch 2/300
6/6 [==============================] - 1s 240ms/step - loss: 4.5307 - mse:
0.1178 - mae: 0.3395 - mape: 95.6391 - val_loss: 4.5461 - val_mse: 0.1905 -
val_mae: 0.4295 - val_mape: 98.9805
```

```
Epoch 3/300
6/6 [==============================] - 2s 255ms/step - loss: 4.4275 - mse:
0.1087 - mae: 0.3258 - mape: 91.7045 - val_loss: 4.4446 - val_mse: 0.1822 -
val_mae: 0.4196 - val_mape: 96.6072
Epoch 4/300
6/6 [==============================] - 1s 171ms/step - loss: 4.3262 - mse:
0.0998 - mae: 0.3117 - mape: 87.6486 - val_loss: 4.3443 - val_mse: 0.1731 -
val_mae: 0.4085 - val_mape: 93.9666
Epoch 5/300
6/6 [==============================] - 1s 80ms/step - loss: 4.2264 - mse: 0.0905
- mae: 0.2962 - mape: 83.2039 - val_loss: 4.2451 - val_mse: 0.1631 - val_mae:
0.3959 - val_mape: 90.9606
Epoch 6/300
6/6 [==============================] - 0s 76ms/step - loss: 4.1279 - mse: 0.0804
- mae: 0.2785 - mape: 78.1337 - val_loss: 4.1466 - val_mse: 0.1517 - val_mae:
0.3812 - val_mape: 87.4493
Epoch 7/300
6/6 [==============================] - 0s 83ms/step - loss: 4.0306 - mse: 0.0694
- mae: 0.2578 - mape: 72.1889 - val_loss: 4.0486 - val_mse: 0.1388 - val_mae:
0.3636 - val_mape: 83.2680
Epoch 8/300
6/6 [==============================] - 0s 77ms/step - loss: 3.9344 - mse: 0.0575
- mae: 0.2332 - mape: 65.1256 - val_loss: 3.9509 - val_mse: 0.1241 - val_mae:
0.3425 - val_mape: 78.2468
Epoch 9/300
6/6 [==============================] - 1s 91ms/step - loss: 3.8396 - mse: 0.0449
- mae: 0.2040 - mape: 56.7494 - val_loss: 3.8535 - val_mse: 0.1077 - val_mae:
0.3172 - val_mape: 72.2370
Epoch 10/300
6/6 [==============================] - 1s 119ms/step - loss: 3.7468 - mse:
0.0324 - mae: 0.1698 - mape: 46.9732 - val_loss: 3.7566 - val_mse: 0.0900 -
val_mae: 0.2875 - val_mape: 65.1844
Epoch 11/300
6/6 [==============================] - 1s 162ms/step - loss: 3.6571 - mse:
0.0212 - mae: 0.1314 - mape: 35.9807 - val_loss: 3.6611 - val_mse: 0.0721 -
val_mae: 0.2538 - val_mape: 57.2257
Epoch 12/300
6/6 [==============================] - 1s 205ms/step - loss: 3.5715 - mse:
0.0126 - mae: 0.0934 - mape: 25.2337 - val_loss: 3.5682 - val_mse: 0.0555 -
val_mae: 0.2182 - val_mape: 48.8167
Epoch 13/300
6/6 [==============================] - 1s 211ms/step - loss: 3.4906 - mse:
0.0076 - mae: 0.0663 - mape: 17.9153 - val_loss: 3.4794 - val_mse: 0.0420 -
val_mae: 0.1842 - val_mape: 40.8086
Epoch 14/300
6/6 [==============================] - 1s 216ms/step - loss: 3.4135 - mse:
0.0055 - mae: 0.0562 - mape: 15.4875 - val_loss: 3.3952 - val_mse: 0.0321 -
val_mae: 0.1552 - val_mape: 34.0339
```

```
Epoch 15/300
6/6 [==============================] - 1s 200ms/step - loss: 3.3387 - mse:
0.0046 - mae: 0.0526 - mape: 14.6669 - val_loss: 3.3153 - val_mse: 0.0255 -
val_mae: 0.1335 - val_mape: 29.0313
Epoch 16/300
6/6 [==============================] - 1s 163ms/step - loss: 3.2652 - mse:
0.0039 - mae: 0.0488 - mape: 13.5958 - val_loss: 3.2390 - val_mse: 0.0211 -
val_mae: 0.1188 - val_mape: 25.7295
Epoch 17/300
6/6 [==============================] - 1s 220ms/step - loss: 3.1934 - mse:
0.0033 - mae: 0.0448 - mape: 12.4116 - val_loss: 3.1654 - val_mse: 0.0179 -
val_mae: 0.1079 - val_mape: 23.3162
Epoch 18/300
6/6 [==============================] - 1s 222ms/step - loss: 3.1233 - mse:
0.0030 - mae: 0.0422 - mape: 11.6156 - val_loss: 3.0941 - val_mse: 0.0153 -
val_mae: 0.0987 - val_mape: 21.2815
Epoch 19/300
6/6 [==============================] - 1s 231ms/step - loss: 3.0549 - mse:
0.0027 - mae: 0.0405 - mape: 11.1124 - val_loss: 3.0248 - val_mse: 0.0131 -
val_mae: 0.0907 - val_mape: 19.5271
Epoch 20/300
6/6 [==============================] - 1s 220ms/step - loss: 2.9881 - mse:
0.0024 - mae: 0.0389 - mape: 10.6938 - val_loss: 2.9574 - val_mse: 0.0114 -
val_mae: 0.0841 - val_mape: 18.1149
Epoch 21/300
6/6 [==============================] - 1s 230ms/step - loss: 2.9228 - mse:
0.0022 - mae: 0.0371 - mape: 10.2175 - val_loss: 2.8919 - val_mse: 0.0101 -
val_mae: 0.0787 - val_mape: 16.9814
Epoch 22/300
6/6 [==============================] - 1s 220ms/step - loss: 2.8590 - mse:
0.0020 - mae: 0.0355 - mape: 9.8042 - val_loss: 2.8282 - val_mse: 0.0092 -
val_mae: 0.0746 - val_mape: 16.1143
Epoch 23/300
6/6 [==============================] - 1s 216ms/step - loss: 2.7965 - mse:
0.0019 - mae: 0.0343 - mape: 9.4941 - val_loss: 2.7660 - val_mse: 0.0085 -
val_mae: 0.0717 - val_mape: 15.5084
Epoch 24/300
6/6 [==============================] - 1s 220ms/step - loss: 2.7354 - mse:
0.0018 - mae: 0.0335 - mape: 9.2717 - val_loss: 2.7053 - val_mse: 0.0081 -
val_mae: 0.0697 - val_mape: 15.1048
Epoch 25/300
6/6 [==============================] - 1s 223ms/step - loss: 2.6756 - mse:
0.0017 - mae: 0.0329 - mape: 9.1078 - val_loss: 2.6460 - val_mse: 0.0078 -
val_mae: 0.0682 - val_mape: 14.7718
Epoch 26/300
6/6 [==============================] - 1s 227ms/step - loss: 2.6171 - mse:
0.0017 - mae: 0.0324 - mape: 8.9733 - val_loss: 2.5881 - val_mse: 0.0075 -
val_mae: 0.0669 - val_mape: 14.4949
```

```
Epoch 27/300
6/6 [==============================] - 1s 214ms/step - loss: 2.5599 - mse:
0.0016 - mae: 0.0319 - mape: 8.8398 - val_loss: 2.5314 - val_mse: 0.0073 -
val_mae: 0.0658 - val_mape: 14.2498
Epoch 28/300
6/6 [==============================] - 1s 219ms/step - loss: 2.5038 - mse:
0.0016 - mae: 0.0314 - mape: 8.6982 - val_loss: 2.4760 - val_mse: 0.0071 -
val_mae: 0.0647 - val_mape: 14.0260
Epoch 29/300
6/6 [==============================] - 1s 223ms/step - loss: 2.4490 - mse:
0.0015 - mae: 0.0308 - mape: 8.5574 - val_loss: 2.4217 - val_mse: 0.0069 -
val_mae: 0.0637 - val_mape: 13.8152
Epoch 30/300
6/6 [==============================] - 1s 217ms/step - loss: 2.3953 - mse:
0.0015 - mae: 0.0303 - mape: 8.4183 - val_loss: 2.3686 - val_mse: 0.0067 -
val_mae: 0.0628 - val_mape: 13.6149
Epoch 31/300
6/6 [==============================] - 1s 240ms/step - loss: 2.3428 - mse:
0.0014 - mae: 0.0298 - mape: 8.2855 - val_loss: 2.3167 - val_mse: 0.0066 -
val_mae: 0.0619 - val_mape: 13.4236
Epoch 32/300
6/6 [==============================] - 1s 220ms/step - loss: 2.2914 - mse:
0.0014 - mae: 0.0294 - mape: 8.1658 - val_loss: 2.2658 - val_mse: 0.0064 -
val_mae: 0.0610 - val_mape: 13.2396
Epoch 33/300
6/6 [==============================] - 1s 230ms/step - loss: 2.2411 - mse:
0.0013 - mae: 0.0290 - mape: 8.0542 - val_loss: 2.2161 - val_mse: 0.0063 -
val_mae: 0.0602 - val_mape: 13.0587
Epoch 34/300
6/6 [==============================] - 1s 227ms/step - loss: 2.1919 - mse:
0.0013 - mae: 0.0286 - mape: 7.9490 - val_loss: 2.1674 - val_mse: 0.0061 -
val_mae: 0.0594 - val_mape: 12.8801
Epoch 35/300
6/6 [==============================] - 1s 247ms/step - loss: 2.1437 - mse:
0.0013 - mae: 0.0282 - mape: 7.8463 - val_loss: 2.1197 - val_mse: 0.0060 -
val_mae: 0.0585 - val_mape: 12.7054
Epoch 36/300
6/6 [==============================] - 1s 230ms/step - loss: 2.0966 - mse:
0.0012 - mae: 0.0279 - mape: 7.7459 - val_loss: 2.0730 - val_mse: 0.0058 -
val_mae: 0.0577 - val_mape: 12.5350
Epoch 37/300
6/6 [==============================] - 1s 237ms/step - loss: 2.0504 - mse:
0.0012 - mae: 0.0275 - mape: 7.6494 - val_loss: 2.0274 - val_mse: 0.0057 -
val_mae: 0.0570 - val_mape: 12.3739
Epoch 38/300
6/6 [==============================] - 1s 223ms/step - loss: 2.0052 - mse:
0.0012 - mae: 0.0272 - mape: 7.5562 - val_loss: 1.9827 - val_mse: 0.0056 -
val_mae: 0.0562 - val_mape: 12.2192
```

```
Epoch 39/300
6/6 [==============================] - 1s 237ms/step - loss: 1.9610 - mse:
0.0012 - mae: 0.0268 - mape: 7.4661 - val_loss: 1.9389 - val_mse: 0.0054 -
val_mae: 0.0555 - val_mape: 12.0677
Epoch 40/300
6/6 [==============================] - 1s 227ms/step - loss: 1.9177 - mse:
0.0011 - mae: 0.0265 - mape: 7.3779 - val_loss: 1.8961 - val_mse: 0.0053 -
val_mae: 0.0548 - val_mape: 11.9195
Epoch 41/300
6/6 [==============================] - 1s 227ms/step - loss: 1.8754 - mse:
0.0011 - mae: 0.0262 - mape: 7.2914 - val_loss: 1.8542 - val_mse: 0.0052 -
val_mae: 0.0541 - val_mape: 11.7749
Epoch 42/300
6/6 [==============================] - 1s 217ms/step - loss: 1.8339 - mse:
0.0011 - mae: 0.0259 - mape: 7.2066 - val_loss: 1.8132 - val_mse: 0.0051 -
val_mae: 0.0535 - val_mape: 11.6338
Epoch 43/300
6/6 [==============================] - 1s 243ms/step - loss: 1.7933 - mse:
0.0011 - mae: 0.0256 - mape: 7.1234 - val_loss: 1.7730 - val_mse: 0.0050 -
val_mae: 0.0528 - val_mape: 11.4963
Epoch 44/300
6/6 [==============================] - 1s 233ms/step - loss: 1.7536 - mse:
0.0010 - mae: 0.0253 - mape: 7.0419 - val_loss: 1.7337 - val_mse: 0.0049 -
val_mae: 0.0522 - val_mape: 11.3629
Epoch 45/300
6/6 [==============================] - 1s 223ms/step - loss: 1.7147 - mse:
0.0010 - mae: 0.0250 - mape: 6.9638 - val_loss: 1.6953 - val_mse: 0.0048 -
val_mae: 0.0516 - val_mape: 11.2322
Epoch 46/300
6/6 [==============================] - 1s 227ms/step - loss: 1.6767 - mse:
9.9495e-04 - mae: 0.0248 - mape: 6.8905 - val_loss: 1.6576 - val_mse: 0.0047 -
val_mae: 0.0509 - val_mape: 11.1036
Epoch 47/300
6/6 [==============================] - 1s 227ms/step - loss: 1.6394 - mse:
9.7535e-04 - mae: 0.0245 - mape: 6.8198 - val_loss: 1.6208 - val_mse: 0.0046 -
val_mae: 0.0504 - val_mape: 10.9841
Epoch 48/300
6/6 [==============================] - 1s 228ms/step - loss: 1.6030 - mse:
9.5663e-04 - mae: 0.0242 - mape: 6.7515 - val_loss: 1.5847 - val_mse: 0.0045 -
val_mae: 0.0498 - val_mape: 10.8704
Epoch 49/300
6/6 [==============================] - 1s 224ms/step - loss: 1.5673 - mse:
9.3873e-04 - mae: 0.0240 - mape: 6.6870 - val_loss: 1.5494 - val_mse: 0.0044 -
val_mae: 0.0493 - val_mape: 10.7587
Epoch 50/300
6/6 [==============================] - 1s 208ms/step - loss: 1.5324 - mse:
9.2151e-04 - mae: 0.0238 - mape: 6.6256 - val_loss: 1.5149 - val_mse: 0.0043 -
val_mae: 0.0488 - val_mape: 10.6477
```

```
Epoch 51/300
6/6 [==============================] - 1s 229ms/step - loss: 1.4982 - mse:
9.0491e-04 - mae: 0.0236 - mape: 6.5675 - val_loss: 1.4811 - val_mse: 0.0042 -
val_mae: 0.0483 - val_mape: 10.5380
Epoch 52/300
6/6 [==============================] - 1s 221ms/step - loss: 1.4647 - mse:
8.8896e-04 - mae: 0.0234 - mape: 6.5113 - val_loss: 1.4480 - val_mse: 0.0041 -
val_mae: 0.0478 - val_mape: 10.4301
Epoch 53/300
6/6 [==============================] - 1s 218ms/step - loss: 1.4320 - mse:
8.7348e-04 - mae: 0.0232 - mape: 6.4575 - val_loss: 1.4156 - val_mse: 0.0040 -
val_mae: 0.0473 - val_mape: 10.3248
Epoch 54/300
6/6 [==============================] - 2s 241ms/step - loss: 1.3999 - mse:
8.5854e-04 - mae: 0.0230 - mape: 6.4054 - val_loss: 1.3839 - val_mse: 0.0039 -
val_mae: 0.0468 - val_mape: 10.2200
Epoch 55/300
6/6 [==============================] - 2s 262ms/step - loss: 1.3686 - mse:
8.4410e-04 - mae: 0.0228 - mape: 6.3540 - val_loss: 1.3529 - val_mse: 0.0039 -
val_mae: 0.0463 - val_mape: 10.1154
Epoch 56/300
6/6 [==============================] - 1s 198ms/step - loss: 1.3379 - mse:
8.3010e-04 - mae: 0.0226 - mape: 6.3040 - val_loss: 1.3225 - val_mse: 0.0038 -
val_mae: 0.0458 - val_mape: 10.0121
Epoch 57/300
6/6 [==============================] - 1s 223ms/step - loss: 1.3078 - mse:
8.1651e-04 - mae: 0.0224 - mape: 6.2554 - val_loss: 1.2928 - val_mse: 0.0037 -
val_mae: 0.0453 - val_mape: 9.9105
Epoch 58/300
6/6 [==============================] - 2s 314ms/step - loss: 1.2784 - mse:
8.0345e-04 - mae: 0.0223 - mape: 6.2079 - val_loss: 1.2637 - val_mse: 0.0037 -
val_mae: 0.0449 - val_mape: 9.8113
Epoch 59/300
6/6 [==============================] - 1s 208ms/step - loss: 1.2496 - mse:
7.9076e-04 - mae: 0.0221 - mape: 6.1611 - val_loss: 1.2353 - val_mse: 0.0036 -
val_mae: 0.0444 - val_mape: 9.7148
Epoch 60/300
6/6 [==============================] - 1s 212ms/step - loss: 1.2215 - mse:
7.7840e-04 - mae: 0.0219 - mape: 6.1156 - val_loss: 1.2074 - val_mse: 0.0035 -
val_mae: 0.0440 - val_mape: 9.6192
Epoch 61/300
6/6 [==============================] - 1s 219ms/step - loss: 1.1939 - mse:
7.6639e-04 - mae: 0.0218 - mape: 6.0705 - val_loss: 1.1801 - val_mse: 0.0035 -
val_mae: 0.0435 - val_mape: 9.5246
Epoch 62/300
6/6 [==============================] - 2s 269ms/step - loss: 1.1669 - mse:
7.5479e-04 - mae: 0.0216 - mape: 6.0257 - val_loss: 1.1535 - val_mse: 0.0034 -
val_mae: 0.0431 - val_mape: 9.4305
```

```
Epoch 63/300
6/6 [==============================] - 1s 217ms/step - loss: 1.1405 - mse:
7.4343e-04 - mae: 0.0214 - mape: 5.9808 - val_loss: 1.1274 - val_mse: 0.0033 -
val_mae: 0.0427 - val_mape: 9.3377
Epoch 64/300
6/6 [==============================] - 1s 192ms/step - loss: 1.1147 - mse:
7.3221e-04 - mae: 0.0213 - mape: 5.9359 - val_loss: 1.1018 - val_mse: 0.0033 -
val_mae: 0.0422 - val_mape: 9.2457
Epoch 65/300
6/6 [==============================] - 1s 203ms/step - loss: 1.0894 - mse:
7.2128e-04 - mae: 0.0211 - mape: 5.8916 - val_loss: 1.0768 - val_mse: 0.0032 -
val_mae: 0.0418 - val_mape: 9.1543
Epoch 66/300
6/6 [==============================] - 1s 210ms/step - loss: 1.0646 - mse:
7.1056e-04 - mae: 0.0210 - mape: 5.8475 - val_loss: 1.0523 - val_mse: 0.0032 -
val_mae: 0.0414 - val_mape: 9.0634
Epoch 67/300
6/6 [==============================] - 1s 237ms/step - loss: 1.0404 - mse:
7.0022e-04 - mae: 0.0208 - mape: 5.8039 - val_loss: 1.0284 - val_mse: 0.0031 -
val_mae: 0.0410 - val_mape: 8.9735
Epoch 68/300
6/6 [==============================] - 1s 215ms/step - loss: 1.0167 - mse:
6.9021e-04 - mae: 0.0206 - mape: 5.7612 - val_loss: 1.0050 - val_mse: 0.0031 -
val_mae: 0.0406 - val_mape: 8.8850
Epoch 69/300
6/6 [==============================] - 1s 211ms/step - loss: 0.9936 - mse:
6.8050e-04 - mae: 0.0205 - mape: 5.7196 - val_loss: 0.9820 - val_mse: 0.0030 -
val_mae: 0.0402 - val_mape: 8.7976
Epoch 70/300
6/6 [==============================] - 1s 217ms/step - loss: 0.9709 - mse:
6.7104e-04 - mae: 0.0204 - mape: 5.6792 - val_loss: 0.9596 - val_mse: 0.0030 -
val_mae: 0.0398 - val_mape: 8.7116
Epoch 71/300
6/6 [==============================] - 1s 195ms/step - loss: 0.9487 - mse:
6.6179e-04 - mae: 0.0202 - mape: 5.6400 - val_loss: 0.9377 - val_mse: 0.0029 -
val_mae: 0.0394 - val_mape: 8.6271
Epoch 72/300
6/6 [==============================] - 1s 221ms/step - loss: 0.9269 - mse:
6.5277e-04 - mae: 0.0201 - mape: 5.6011 - val_loss: 0.9162 - val_mse: 0.0029 -
val_mae: 0.0390 - val_mape: 8.5437
Epoch 73/300
6/6 [==============================] - 1s 204ms/step - loss: 0.9057 - mse:
6.4392e-04 - mae: 0.0199 - mape: 5.5643 - val_loss: 0.8952 - val_mse: 0.0028 -
val_mae: 0.0387 - val_mape: 8.4745
Epoch 74/300
6/6 [==============================] - 1s 217ms/step - loss: 0.8849 - mse:
6.3524e-04 - mae: 0.0198 - mape: 5.5287 - val_loss: 0.8746 - val_mse: 0.0028 -
val_mae: 0.0384 - val_mape: 8.4072
```

```
Epoch 75/300
6/6 [==============================] - 1s 213ms/step - loss: 0.8645 - mse:
6.2695e-04 - mae: 0.0197 - mape: 5.4947 - val_loss: 0.8545 - val_mse: 0.0027 -
val_mae: 0.0381 - val_mape: 8.3471
Epoch 76/300
6/6 [==============================] - 1s 204ms/step - loss: 0.8446 - mse:
6.1893e-04 - mae: 0.0196 - mape: 5.4609 - val_loss: 0.8348 - val_mse: 0.0027 -
val_mae: 0.0378 - val_mape: 8.2982
Epoch 77/300
6/6 [==============================] - 1s 204ms/step - loss: 0.8251 - mse:
6.1107e-04 - mae: 0.0195 - mape: 5.4275 - val_loss: 0.8156 - val_mse: 0.0027 -
val_mae: 0.0376 - val_mape: 8.2516
Epoch 78/300
6/6 [==============================] - 1s 210ms/step - loss: 0.8061 - mse:
6.0339e-04 - mae: 0.0193 - mape: 5.3942 - val_loss: 0.7967 - val_mse: 0.0026 -
val_mae: 0.0373 - val_mape: 8.2049
Epoch 79/300
6/6 [==============================] - 1s 218ms/step - loss: 0.7874 - mse:
5.9591e-04 - mae: 0.0192 - mape: 5.3608 - val_loss: 0.7783 - val_mse: 0.0026 -
val_mae: 0.0371 - val_mape: 8.1583
Epoch 80/300
6/6 [==============================] - 1s 207ms/step - loss: 0.7692 - mse:
5.8851e-04 - mae: 0.0191 - mape: 5.3274 - val_loss: 0.7603 - val_mse: 0.0025 -
val_mae: 0.0368 - val_mape: 8.1126
Epoch 81/300
6/6 [==============================] - 2s 256ms/step - loss: 0.7513 - mse:
5.8132e-04 - mae: 0.0190 - mape: 5.2944 - val_loss: 0.7426 - val_mse: 0.0025 -
val_mae: 0.0366 - val_mape: 8.0674
Epoch 82/300
6/6 [==============================] - 1s 202ms/step - loss: 0.7339 - mse:
5.7441e-04 - mae: 0.0189 - mape: 5.2615 - val_loss: 0.7254 - val_mse: 0.0025 -
val_mae: 0.0364 - val_mape: 8.0221
Epoch 83/300
6/6 [==============================] - 1s 200ms/step - loss: 0.7168 - mse:
5.6779e-04 - mae: 0.0187 - mape: 5.2294 - val_loss: 0.7085 - val_mse: 0.0024 -
val_mae: 0.0361 - val_mape: 7.9775
Epoch 84/300
6/6 [==============================] - 1s 220ms/step - loss: 0.7001 - mse:
5.6133e-04 - mae: 0.0186 - mape: 5.1984 - val_loss: 0.6920 - val_mse: 0.0024 -
val_mae: 0.0359 - val_mape: 7.9337
Epoch 85/300
6/6 [==============================] - 1s 217ms/step - loss: 0.6838 - mse:
5.5500e-04 - mae: 0.0185 - mape: 5.1685 - val_loss: 0.6758 - val_mse: 0.0024 -
val_mae: 0.0357 - val_mape: 7.8904
Epoch 86/300
6/6 [==============================] - 1s 216ms/step - loss: 0.6678 - mse:
5.4899e-04 - mae: 0.0184 - mape: 5.1402 - val_loss: 0.6600 - val_mse: 0.0023 -
val_mae: 0.0355 - val_mape: 7.8474
```

```
Epoch 87/300
6/6 [==============================] - 1s 210ms/step - loss: 0.6521 - mse:
5.4318e-04 - mae: 0.0183 - mape: 5.1133 - val_loss: 0.6446 - val_mse: 0.0023 -
val_mae: 0.0353 - val_mape: 7.8049
Epoch 88/300
6/6 [==============================] - 1s 225ms/step - loss: 0.6368 - mse:
5.3750e-04 - mae: 0.0182 - mape: 5.0869 - val_loss: 0.6295 - val_mse: 0.0023 -
val_mae: 0.0351 - val_mape: 7.7636
Epoch 89/300
6/6 [==============================] - 1s 219ms/step - loss: 0.6219 - mse:
5.3194e-04 - mae: 0.0181 - mape: 5.0608 - val_loss: 0.6147 - val_mse: 0.0023 -
val_mae: 0.0348 - val_mape: 7.7237
Epoch 90/300
6/6 [==============================] - 1s 219ms/step - loss: 0.6072 - mse:
5.2658e-04 - mae: 0.0180 - mape: 5.0355 - val_loss: 0.6002 - val_mse: 0.0022 -
val_mae: 0.0346 - val_mape: 7.6838
Epoch 91/300
6/6 [==============================] - 1s 214ms/step - loss: 0.5929 - mse:
5.2140e-04 - mae: 0.0180 - mape: 5.0108 - val_loss: 0.5861 - val_mse: 0.0022 -
val_mae: 0.0344 - val_mape: 7.6447
Epoch 92/300
6/6 [==============================] - 1s 213ms/step - loss: 0.5789 - mse:
5.1640e-04 - mae: 0.0179 - mape: 4.9873 - val_loss: 0.5723 - val_mse: 0.0022 -
val_mae: 0.0343 - val_mape: 7.6061
Epoch 93/300
6/6 [==============================] - 1s 242ms/step - loss: 0.5653 - mse:
5.1155e-04 - mae: 0.0178 - mape: 4.9643 - val_loss: 0.5587 - val_mse: 0.0022 -
val_mae: 0.0341 - val_mape: 7.5683
Epoch 94/300
6/6 [==============================] - 1s 205ms/step - loss: 0.5519 - mse:
5.0683e-04 - mae: 0.0177 - mape: 4.9414 - val_loss: 0.5455 - val_mse: 0.0021 -
val_mae: 0.0339 - val_mape: 7.5312
Epoch 95/300
6/6 [==============================] - 1s 234ms/step - loss: 0.5388 - mse:
5.0220e-04 - mae: 0.0176 - mape: 4.9188 - val_loss: 0.5326 - val_mse: 0.0021 -
val_mae: 0.0337 - val_mape: 7.4953
Epoch 96/300
6/6 [==============================] - 1s 228ms/step - loss: 0.5260 - mse:
4.9776e-04 - mae: 0.0175 - mape: 4.8966 - val_loss: 0.5199 - val_mse: 0.0021 -
val_mae: 0.0335 - val_mape: 7.4599
Epoch 97/300
6/6 [==============================] - 1s 231ms/step - loss: 0.5135 - mse:
4.9345e-04 - mae: 0.0175 - mape: 4.8747 - val_loss: 0.5076 - val_mse: 0.0021 -
val_mae: 0.0333 - val_mape: 7.4250
Epoch 98/300
6/6 [==============================] - 1s 235ms/step - loss: 0.5012 - mse:
4.8920e-04 - mae: 0.0174 - mape: 4.8530 - val_loss: 0.4955 - val_mse: 0.0020 -
val_mae: 0.0332 - val_mape: 7.3911
```

```
Epoch 99/300
6/6 [==============================] - 1s 205ms/step - loss: 0.4892 - mse:
4.8494e-04 - mae: 0.0173 - mape: 4.8312 - val_loss: 0.4836 - val_mse: 0.0020 -
val_mae: 0.0330 - val_mape: 7.3582
Epoch 100/300
6/6 [==============================] - 1s 211ms/step - loss: 0.4775 - mse:
4.8076e-04 - mae: 0.0172 - mape: 4.8097 - val_loss: 0.4721 - val_mse: 0.0020 -
val_mae: 0.0329 - val_mape: 7.3250
Epoch 101/300
6/6 [==============================] - 1s 216ms/step - loss: 0.4661 - mse:
4.7682e-04 - mae: 0.0172 - mape: 4.7893 - val_loss: 0.4608 - val_mse: 0.0020 -
val_mae: 0.0327 - val_mape: 7.2919
Epoch 102/300
6/6 [==============================] - 1s 202ms/step - loss: 0.4549 - mse:
4.7309e-04 - mae: 0.0171 - mape: 4.7698 - val_loss: 0.4497 - val_mse: 0.0020 -
val_mae: 0.0325 - val_mape: 7.2592
Epoch 103/300
6/6 [==============================] - 1s 220ms/step - loss: 0.4440 - mse:
4.6944e-04 - mae: 0.0170 - mape: 4.7503 - val_loss: 0.4389 - val_mse: 0.0019 -
val_mae: 0.0324 - val_mape: 7.2268
Epoch 104/300
6/6 [==============================] - 1s 213ms/step - loss: 0.4333 - mse:
4.6580e-04 - mae: 0.0169 - mape: 4.7311 - val_loss: 0.4284 - val_mse: 0.0019 -
val_mae: 0.0322 - val_mape: 7.1953
Epoch 105/300
6/6 [==============================] - 1s 207ms/step - loss: 0.4228 - mse:
4.6227e-04 - mae: 0.0169 - mape: 4.7123 - val_loss: 0.4180 - val_mse: 0.0019 -
val_mae: 0.0321 - val_mape: 7.1634
Epoch 106/300
6/6 [==============================] - 1s 217ms/step - loss: 0.4126 - mse:
4.5887e-04 - mae: 0.0168 - mape: 4.6948 - val_loss: 0.4079 - val_mse: 0.0019 -
val_mae: 0.0319 - val_mape: 7.1323
Epoch 107/300
6/6 [==============================] - 1s 223ms/step - loss: 0.4026 - mse:
4.5557e-04 - mae: 0.0167 - mape: 4.6775 - val_loss: 0.3981 - val_mse: 0.0019 -
val_mae: 0.0318 - val_mape: 7.1019
Epoch 108/300
6/6 [==============================] - 2s 244ms/step - loss: 0.3928 - mse:
4.5239e-04 - mae: 0.0167 - mape: 4.6603 - val_loss: 0.3884 - val_mse: 0.0019 -
val_mae: 0.0316 - val_mape: 7.0720
Epoch 109/300
6/6 [==============================] - 2s 252ms/step - loss: 0.3833 - mse:
4.4931e-04 - mae: 0.0166 - mape: 4.6434 - val_loss: 0.3790 - val_mse: 0.0018 -
val_mae: 0.0315 - val_mape: 7.0421
Epoch 110/300
6/6 [==============================] - 1s 236ms/step - loss: 0.3739 - mse:
4.4625e-04 - mae: 0.0166 - mape: 4.6275 - val_loss: 0.3698 - val_mse: 0.0018 -
val_mae: 0.0313 - val_mape: 7.0124
```

```
Epoch 111/300
6/6 [==============================] - 2s 288ms/step - loss: 0.3648 - mse:
4.4321e-04 - mae: 0.0165 - mape: 4.6114 - val_loss: 0.3608 - val_mse: 0.0018 -
val_mae: 0.0312 - val_mape: 6.9838
Epoch 112/300
6/6 [==============================] - 1s 218ms/step - loss: 0.3559 - mse:
4.4030e-04 - mae: 0.0164 - mape: 4.5966 - val_loss: 0.3520 - val_mse: 0.0018 -
val_mae: 0.0310 - val_mape: 6.9554
Epoch 113/300
6/6 [==============================] - 1s 219ms/step - loss: 0.3472 - mse:
4.3761e-04 - mae: 0.0164 - mape: 4.5820 - val_loss: 0.3434 - val_mse: 0.0018 -
val_mae: 0.0309 - val_mape: 6.9297
Epoch 114/300
6/6 [==============================] - 1s 231ms/step - loss: 0.3387 - mse:
4.3493e-04 - mae: 0.0163 - mape: 4.5678 - val_loss: 0.3350 - val_mse: 0.0018 -
val_mae: 0.0308 - val_mape: 6.9050
Epoch 115/300
6/6 [==============================] - 2s 261ms/step - loss: 0.3304 - mse:
4.3233e-04 - mae: 0.0163 - mape: 4.5541 - val_loss: 0.3267 - val_mse: 0.0018 -
val_mae: 0.0307 - val_mape: 6.8811
Epoch 116/300
6/6 [==============================] - 1s 242ms/step - loss: 0.3222 - mse:
4.2982e-04 - mae: 0.0162 - mape: 4.5406 - val_loss: 0.3187 - val_mse: 0.0018 -
val_mae: 0.0306 - val_mape: 6.8571
Epoch 117/300
6/6 [==============================] - 1s 214ms/step - loss: 0.3143 - mse:
4.2730e-04 - mae: 0.0162 - mape: 4.5272 - val_loss: 0.3109 - val_mse: 0.0017 -
val_mae: 0.0305 - val_mape: 6.8336
Epoch 118/300
6/6 [==============================] - 1s 225ms/step - loss: 0.3065 - mse:
4.2484e-04 - mae: 0.0161 - mape: 4.5138 - val_loss: 0.3032 - val_mse: 0.0017 -
val_mae: 0.0303 - val_mape: 6.8102
Epoch 119/300
6/6 [==============================] - 1s 206ms/step - loss: 0.2989 - mse:
4.2254e-04 - mae: 0.0161 - mape: 4.5007 - val_loss: 0.2957 - val_mse: 0.0017 -
val_mae: 0.0302 - val_mape: 6.7869
Epoch 120/300
6/6 [==============================] - 1s 206ms/step - loss: 0.2915 - mse:
4.2032e-04 - mae: 0.0160 - mape: 4.4888 - val_loss: 0.2884 - val_mse: 0.0017 -
val_mae: 0.0301 - val_mape: 6.7651
Epoch 121/300
6/6 [==============================] - 1s 210ms/step - loss: 0.2843 - mse:
4.1812e-04 - mae: 0.0160 - mape: 4.4785 - val_loss: 0.2812 - val_mse: 0.0017 -
val_mae: 0.0300 - val_mape: 6.7445
Epoch 122/300
6/6 [==============================] - 1s 223ms/step - loss: 0.2772 - mse:
4.1607e-04 - mae: 0.0160 - mape: 4.4694 - val_loss: 0.2743 - val_mse: 0.0017 -
val_mae: 0.0299 - val_mape: 6.7248
```

```
Epoch 123/300
6/6 [==============================] - 2s 252ms/step - loss: 0.2703 - mse:
4.1423e-04 - mae: 0.0159 - mape: 4.4614 - val_loss: 0.2674 - val_mse: 0.0017 -
val_mae: 0.0298 - val_mape: 6.7063
Epoch 124/300
6/6 [==============================] - 1s 250ms/step - loss: 0.2635 - mse:
4.1247e-04 - mae: 0.0159 - mape: 4.4550 - val_loss: 0.2608 - val_mse: 0.0017 -
val_mae: 0.0298 - val_mape: 6.6888
Epoch 125/300
6/6 [==============================] - 1s 229ms/step - loss: 0.2569 - mse:
4.1076e-04 - mae: 0.0159 - mape: 4.4488 - val_loss: 0.2542 - val_mse: 0.0017 -
val_mae: 0.0297 - val_mape: 6.6717
Epoch 126/300
6/6 [==============================] - 1s 216ms/step - loss: 0.2505 - mse:
4.0911e-04 - mae: 0.0159 - mape: 4.4422 - val_loss: 0.2479 - val_mse: 0.0017 -
val_mae: 0.0296 - val_mape: 6.6546
Epoch 127/300
6/6 [==============================] - 1s 239ms/step - loss: 0.2442 - mse:
4.0757e-04 - mae: 0.0158 - mape: 4.4359 - val_loss: 0.2417 - val_mse: 0.0016 -
val_mae: 0.0295 - val_mape: 6.6381
Epoch 128/300
6/6 [==============================] - 1s 224ms/step - loss: 0.2380 - mse:
4.0609e-04 - mae: 0.0158 - mape: 4.4305 - val_loss: 0.2356 - val_mse: 0.0016 -
val_mae: 0.0294 - val_mape: 6.6218
Epoch 129/300
6/6 [==============================] - 1s 227ms/step - loss: 0.2320 - mse:
4.0467e-04 - mae: 0.0158 - mape: 4.4259 - val_loss: 0.2297 - val_mse: 0.0016 -
val_mae: 0.0293 - val_mape: 6.6057
Epoch 130/300
6/6 [==============================] - 1s 233ms/step - loss: 0.2262 - mse:
4.0331e-04 - mae: 0.0158 - mape: 4.4222 - val_loss: 0.2239 - val_mse: 0.0016 -
val_mae: 0.0293 - val_mape: 6.5902
Epoch 131/300
6/6 [==============================] - 1s 237ms/step - loss: 0.2204 - mse:
4.0203e-04 - mae: 0.0158 - mape: 4.4188 - val_loss: 0.2182 - val_mse: 0.0016 -
val_mae: 0.0292 - val_mape: 6.5746
Epoch 132/300
6/6 [==============================] - 1s 233ms/step - loss: 0.2148 - mse:
4.0082e-04 - mae: 0.0157 - mape: 4.4157 - val_loss: 0.2127 - val_mse: 0.0016 -
val_mae: 0.0291 - val_mape: 6.5589
Epoch 133/300
6/6 [==============================] - 1s 230ms/step - loss: 0.2094 - mse:
3.9968e-04 - mae: 0.0157 - mape: 4.4129 - val_loss: 0.2073 - val_mse: 0.0016 -
val_mae: 0.0290 - val_mape: 6.5434
Epoch 134/300
6/6 [==============================] - 1s 227ms/step - loss: 0.2040 - mse:
3.9857e-04 - mae: 0.0157 - mape: 4.4101 - val_loss: 0.2021 - val_mse: 0.0016 -
val_mae: 0.0290 - val_mape: 6.5283
```

```
Epoch 135/300
6/6 [==============================] - 1s 233ms/step - loss: 0.1988 - mse:
3.9752e-04 - mae: 0.0157 - mape: 4.4075 - val_loss: 0.1969 - val_mse: 0.0016 -
val_mae: 0.0289 - val_mape: 6.5159
Epoch 136/300
6/6 [==============================] - 1s 240ms/step - loss: 0.1938 - mse:
3.9660e-04 - mae: 0.0157 - mape: 4.4059 - val_loss: 0.1919 - val_mse: 0.0016 -
val_mae: 0.0289 - val_mape: 6.5049
Epoch 137/300
6/6 [==============================] - 1s 230ms/step - loss: 0.1888 - mse:
3.9577e-04 - mae: 0.0157 - mape: 4.4043 - val_loss: 0.1870 - val_mse: 0.0016 -
val_mae: 0.0288 - val_mape: 6.4938
Epoch 138/300
6/6 [==============================] - 1s 217ms/step - loss: 0.1839 - mse:
3.9500e-04 - mae: 0.0157 - mape: 4.4033 - val_loss: 0.1822 - val_mse: 0.0016 -
val_mae: 0.0288 - val_mape: 6.4833
Epoch 139/300
6/6 [==============================] - 1s 234ms/step - loss: 0.1792 - mse:
3.9429e-04 - mae: 0.0157 - mape: 4.4031 - val_loss: 0.1776 - val_mse: 0.0016 -
val_mae: 0.0287 - val_mape: 6.4740
Epoch 140/300
6/6 [==============================] - 1s 227ms/step - loss: 0.1746 - mse:
3.9365e-04 - mae: 0.0157 - mape: 4.4028 - val_loss: 0.1730 - val_mse: 0.0016 -
val_mae: 0.0287 - val_mape: 6.4654
Epoch 141/300
6/6 [==============================] - 1s 233ms/step - loss: 0.1701 - mse:
3.9314e-04 - mae: 0.0157 - mape: 4.4024 - val_loss: 0.1686 - val_mse: 0.0016 -
val_mae: 0.0286 - val_mape: 6.4573
Epoch 142/300
6/6 [==============================] - 1s 231ms/step - loss: 0.1657 - mse:
3.9269e-04 - mae: 0.0157 - mape: 4.4026 - val_loss: 0.1642 - val_mse: 0.0016 -
val_mae: 0.0286 - val_mape: 6.4500
Epoch 143/300
6/6 [==============================] - 1s 227ms/step - loss: 0.1614 - mse:
3.9224e-04 - mae: 0.0157 - mape: 4.4031 - val_loss: 0.1600 - val_mse: 0.0016 -
val_mae: 0.0286 - val_mape: 6.4434
Epoch 144/300
6/6 [==============================] - 1s 233ms/step - loss: 0.1572 - mse:
3.9182e-04 - mae: 0.0157 - mape: 4.4037 - val_loss: 0.1559 - val_mse: 0.0016 -
val_mae: 0.0285 - val_mape: 6.4369
Epoch 145/300
6/6 [==============================] - 1s 227ms/step - loss: 0.1531 - mse:
3.9152e-04 - mae: 0.0157 - mape: 4.4040 - val_loss: 0.1518 - val_mse: 0.0016 -
val_mae: 0.0285 - val_mape: 6.4316
Epoch 146/300
6/6 [==============================] - 1s 221ms/step - loss: 0.1491 - mse:
3.9130e-04 - mae: 0.0157 - mape: 4.4045 - val_loss: 0.1479 - val_mse: 0.0015 -
val_mae: 0.0285 - val_mape: 6.4269
```

```
Epoch 147/300
6/6 [==============================] - 1s 233ms/step - loss: 0.1452 - mse:
3.9110e-04 - mae: 0.0157 - mape: 4.4061 - val_loss: 0.1440 - val_mse: 0.0015 -
val_mae: 0.0284 - val_mape: 6.4218
Epoch 148/300
6/6 [==============================] - 1s 236ms/step - loss: 0.1414 - mse:
3.9094e-04 - mae: 0.0157 - mape: 4.4076 - val_loss: 0.1403 - val_mse: 0.0015 -
val_mae: 0.0284 - val_mape: 6.4170
Epoch 149/300
6/6 [==============================] - 1s 230ms/step - loss: 0.1377 - mse:
3.9089e-04 - mae: 0.0157 - mape: 4.4092 - val_loss: 0.1366 - val_mse: 0.0015 -
val_mae: 0.0284 - val_mape: 6.4126
Epoch 150/300
6/6 [==============================] - 1s 230ms/step - loss: 0.1341 - mse:
3.9088e-04 - mae: 0.0157 - mape: 4.4104 - val_loss: 0.1331 - val_mse: 0.0015 -
val_mae: 0.0284 - val_mape: 6.4080
Epoch 151/300
6/6 [==============================] - 1s 243ms/step - loss: 0.1305 - mse:
3.9093e-04 - mae: 0.0157 - mape: 4.4109 - val_loss: 0.1296 - val_mse: 0.0015 -
val_mae: 0.0284 - val_mape: 6.4033
Epoch 152/300
6/6 [==============================] - 1s 248ms/step - loss: 0.1271 - mse:
3.9100e-04 - mae: 0.0157 - mape: 4.4115 - val_loss: 0.1262 - val_mse: 0.0015 -
val_mae: 0.0283 - val_mape: 6.3994
Epoch 153/300
6/6 [==============================] - 1s 214ms/step - loss: 0.1237 - mse:
3.9113e-04 - mae: 0.0157 - mape: 4.4133 - val_loss: 0.1229 - val_mse: 0.0015 -
val_mae: 0.0283 - val_mape: 6.3963
Epoch 154/300
6/6 [==============================] - 1s 208ms/step - loss: 0.1205 - mse:
3.9130e-04 - mae: 0.0157 - mape: 4.4156 - val_loss: 0.1197 - val_mse: 0.0015 -
val_mae: 0.0283 - val_mape: 6.3933
Epoch 155/300
6/6 [==============================] - 1s 211ms/step - loss: 0.1173 - mse:
3.9160e-04 - mae: 0.0157 - mape: 4.4177 - val_loss: 0.1165 - val_mse: 0.0015 -
val_mae: 0.0283 - val_mape: 6.3909
Epoch 156/300
6/6 [==============================] - 1s 226ms/step - loss: 0.1141 - mse:
3.9199e-04 - mae: 0.0157 - mape: 4.4199 - val_loss: 0.1134 - val_mse: 0.0015 -
val_mae: 0.0283 - val_mape: 6.3890
Epoch 157/300
6/6 [==============================] - 1s 206ms/step - loss: 0.1111 - mse:
3.9239e-04 - mae: 0.0157 - mape: 4.4227 - val_loss: 0.1104 - val_mse: 0.0015 -
val_mae: 0.0283 - val_mape: 6.3868
Epoch 158/300
6/6 [==============================] - 1s 216ms/step - loss: 0.1081 - mse:
3.9279e-04 - mae: 0.0157 - mape: 4.4254 - val_loss: 0.1075 - val_mse: 0.0015 -
val_mae: 0.0283 - val_mape: 6.3848
```

```
Epoch 159/300
6/6 [==============================] - 1s 210ms/step - loss: 0.1052 - mse:
3.9322e-04 - mae: 0.0157 - mape: 4.4281 - val_loss: 0.1047 - val_mse: 0.0015 -
val_mae: 0.0283 - val_mape: 6.3834
Epoch 160/300
6/6 [==============================] - 1s 209ms/step - loss: 0.1024 - mse:
3.9366e-04 - mae: 0.0157 - mape: 4.4312 - val_loss: 0.1019 - val_mse: 0.0015 -
val_mae: 0.0282 - val_mape: 6.3816
Epoch 161/300
6/6 [==============================] - 1s 206ms/step - loss: 0.0997 - mse:
3.9414e-04 - mae: 0.0157 - mape: 4.4345 - val_loss: 0.0992 - val_mse: 0.0015 -
val_mae: 0.0282 - val_mape: 6.3800
Epoch 162/300
6/6 [==============================] - 1s 200ms/step - loss: 0.0970 - mse:
3.9466e-04 - mae: 0.0157 - mape: 4.4382 - val_loss: 0.0965 - val_mse: 0.0015 -
val_mae: 0.0282 - val_mape: 6.3815
Epoch 163/300
6/6 [==============================] - 1s 218ms/step - loss: 0.0944 - mse:
3.9524e-04 - mae: 0.0158 - mape: 4.4418 - val_loss: 0.0940 - val_mse: 0.0015 -
val_mae: 0.0282 - val_mape: 6.3840
Epoch 164/300
6/6 [==============================] - 1s 221ms/step - loss: 0.0918 - mse:
3.9588e-04 - mae: 0.0158 - mape: 4.4454 - val_loss: 0.0914 - val_mse: 0.0015 -
val_mae: 0.0283 - val_mape: 6.3857
Epoch 165/300
6/6 [==============================] - 1s 219ms/step - loss: 0.0893 - mse:
3.9657e-04 - mae: 0.0158 - mape: 4.4492 - val_loss: 0.0890 - val_mse: 0.0015 -
val_mae: 0.0283 - val_mape: 6.3880
Epoch 166/300
6/6 [==============================] - 1s 229ms/step - loss: 0.0869 - mse:
3.9730e-04 - mae: 0.0158 - mape: 4.4539 - val_loss: 0.0866 - val_mse: 0.0015 -
val_mae: 0.0283 - val_mape: 6.3911
Epoch 167/300
6/6 [==============================] - 1s 229ms/step - loss: 0.0846 - mse:
3.9808e-04 - mae: 0.0158 - mape: 4.4590 - val_loss: 0.0843 - val_mse: 0.0015 -
val_mae: 0.0283 - val_mape: 6.3952
Epoch 168/300
6/6 [==============================] - 1s 241ms/step - loss: 0.0823 - mse:
3.9894e-04 - mae: 0.0158 - mape: 4.4637 - val_loss: 0.0820 - val_mse: 0.0015 -
val_mae: 0.0283 - val_mape: 6.3999
Epoch 169/300
6/6 [==============================] - 1s 215ms/step - loss: 0.0800 - mse:
3.9991e-04 - mae: 0.0158 - mape: 4.4687 - val_loss: 0.0798 - val_mse: 0.0015 -
val_mae: 0.0283 - val_mape: 6.4050
Epoch 170/300
6/6 [==============================] - 1s 204ms/step - loss: 0.0778 - mse:
4.0091e-04 - mae: 0.0159 - mape: 4.4746 - val_loss: 0.0777 - val_mse: 0.0015 -
val_mae: 0.0283 - val_mape: 6.4105
```

```
Epoch 171/300
6/6 [==============================] - 1s 239ms/step - loss: 0.0757 - mse:
4.0191e-04 - mae: 0.0159 - mape: 4.4813 - val_loss: 0.0756 - val_mse: 0.0015 -
val_mae: 0.0284 - val_mape: 6.4163
Epoch 172/300
6/6 [==============================] - 1s 219ms/step - loss: 0.0736 - mse:
4.0294e-04 - mae: 0.0159 - mape: 4.4871 - val_loss: 0.0735 - val_mse: 0.0015 -
val_mae: 0.0284 - val_mape: 6.4224
Epoch 173/300
6/6 [==============================] - 1s 200ms/step - loss: 0.0716 - mse:
4.0401e-04 - mae: 0.0159 - mape: 4.4928 - val_loss: 0.0715 - val_mse: 0.0015 -
val_mae: 0.0284 - val_mape: 6.4285
Epoch 174/300
6/6 [==============================] - 1s 218ms/step - loss: 0.0696 - mse:
4.0510e-04 - mae: 0.0159 - mape: 4.4992 - val_loss: 0.0696 - val_mse: 0.0015 -
val_mae: 0.0284 - val_mape: 6.4345
Epoch 175/300
6/6 [==============================] - 2s 316ms/step - loss: 0.0677 - mse:
4.0617e-04 - mae: 0.0160 - mape: 4.5060 - val_loss: 0.0677 - val_mse: 0.0015 -
val_mae: 0.0285 - val_mape: 6.4407
Epoch 176/300
6/6 [==============================] - 2s 268ms/step - loss: 0.0658 - mse:
4.0729e-04 - mae: 0.0160 - mape: 4.5128 - val_loss: 0.0659 - val_mse: 0.0015 -
val_mae: 0.0285 - val_mape: 6.4479
Epoch 177/300
6/6 [==============================] - 1s 219ms/step - loss: 0.0640 - mse:
4.0851e-04 - mae: 0.0160 - mape: 4.5197 - val_loss: 0.0641 - val_mse: 0.0015 -
val_mae: 0.0285 - val_mape: 6.4553
Epoch 178/300
6/6 [==============================] - 1s 236ms/step - loss: 0.0622 - mse:
4.0974e-04 - mae: 0.0160 - mape: 4.5267 - val_loss: 0.0623 - val_mse: 0.0016 -
val_mae: 0.0286 - val_mape: 6.4627
Epoch 179/300
6/6 [==============================] - 1s 228ms/step - loss: 0.0605 - mse:
4.1102e-04 - mae: 0.0160 - mape: 4.5340 - val_loss: 0.0606 - val_mse: 0.0016 -
val_mae: 0.0286 - val_mape: 6.4707
Epoch 180/300
6/6 [==============================] - 1s 233ms/step - loss: 0.0588 - mse:
4.1238e-04 - mae: 0.0161 - mape: 4.5418 - val_loss: 0.0590 - val_mse: 0.0016 -
val_mae: 0.0286 - val_mape: 6.4785
Epoch 181/300
6/6 [==============================] - 1s 210ms/step - loss: 0.0572 - mse:
4.1370e-04 - mae: 0.0161 - mape: 4.5501 - val_loss: 0.0574 - val_mse: 0.0016 -
val_mae: 0.0287 - val_mape: 6.4868
Epoch 182/300
6/6 [==============================] - 1s 215ms/step - loss: 0.0556 - mse:
4.1509e-04 - mae: 0.0161 - mape: 4.5586 - val_loss: 0.0558 - val_mse: 0.0016 -
val_mae: 0.0287 - val_mape: 6.4946
```

```
Epoch 183/300
6/6 [==============================] - 1s 219ms/step - loss: 0.0540 - mse:
4.1643e-04 - mae: 0.0161 - mape: 4.5664 - val_loss: 0.0543 - val_mse: 0.0016 -
val_mae: 0.0287 - val_mape: 6.5024
Epoch 184/300
6/6 [==============================] - 1s 214ms/step - loss: 0.0525 - mse:
4.1778e-04 - mae: 0.0162 - mape: 4.5736 - val_loss: 0.0528 - val_mse: 0.0016 -
val_mae: 0.0288 - val_mape: 6.5101
Epoch 185/300
6/6 [==============================] - 1s 210ms/step - loss: 0.0510 - mse:
4.1916e-04 - mae: 0.0162 - mape: 4.5813 - val_loss: 0.0513 - val_mse: 0.0016 -
val_mae: 0.0288 - val_mape: 6.5180
Epoch 186/300
6/6 [==============================] - 1s 226ms/step - loss: 0.0496 - mse:
4.2055e-04 - mae: 0.0162 - mape: 4.5898 - val_loss: 0.0499 - val_mse: 0.0016 -
val_mae: 0.0288 - val_mape: 6.5266
Model: "TCN"
________________________________________________________________________________
__________________
 Layer (type)                   Output Shape         Param #     Connected to
================================================================================
==================
 input_4 (InputLayer)           [(None, 12, 7)]      0           []

 Conv1D_1_0 (Conv1D)            (None, 12, 128)      3584
['input_4[0][0]']

 SpatialDropout1D_1_0 (SpatialD  (None, 12, 128)     0
['Conv1D_1_0[0][0]']
 ropout1D)

 Conv1D_2_0 (Conv1D)            (None, 12, 128)      65536
['SpatialDropout1D_1_0[0][0]']

 SpatialDropout1D_2_0 (SpatialD  (None, 12, 128)     0
['Conv1D_2_0[0][0]']
 ropout1D)

 Conv1D_skipconnection_0 (Conv1  (None, 12, 128)     1024
['input_4[0][0]']
 D)

 residual_Add_0 (Add)           (None, 12, 128)      0
['SpatialDropout1D_2_0[0][0]',
'Conv1D_skipconnection_0[0][0]']

 Conv1D_1_1 (Conv1D)            (None, 12, 128)      65536
['residual_Add_0[0][0]']
```

```
 SpatialDropout1D_1_1 (SpatialD  (None, 12, 128)     0
['Conv1D_1_1[0][0]']
 ropout1D)

 Conv1D_2_1 (Conv1D)            (None, 12, 128)      65536
['SpatialDropout1D_1_1[0][0]']

 SpatialDropout1D_2_1 (SpatialD  (None, 12, 128)     0
['Conv1D_2_1[0][0]']
 ropout1D)

 Conv1D_skipconnection_1 (Conv1  (None, 12, 128)     16512
['residual_Add_0[0][0]']
 D)

 residual_Add_1 (Add)           (None, 12, 128)      0
['SpatialDropout1D_2_1[0][0]',
'Conv1D_skipconnection_1[0][0]']

 Conv1D_1_2 (Conv1D)            (None, 12, 128)      65536
['residual_Add_1[0][0]']

 SpatialDropout1D_1_2 (SpatialD  (None, 12, 128)     0
['Conv1D_1_2[0][0]']
 ropout1D)

 Conv1D_2_2 (Conv1D)            (None, 12, 128)      65536
['SpatialDropout1D_1_2[0][0]']

 SpatialDropout1D_2_2 (SpatialD  (None, 12, 128)     0
['Conv1D_2_2[0][0]']
 ropout1D)

 Conv1D_skipconnection_2 (Conv1  (None, 12, 128)     16512
['residual_Add_1[0][0]']
 D)

 residual_Add_2 (Add)           (None, 12, 128)      0
['SpatialDropout1D_2_2[0][0]',
'Conv1D_skipconnection_2[0][0]']

 Conv1D_1_3 (Conv1D)            (None, 12, 128)      65536
['residual_Add_2[0][0]']

 SpatialDropout1D_1_3 (SpatialD  (None, 12, 128)     0
['Conv1D_1_3[0][0]']
 ropout1D)
```

```
 Conv1D_2_3 (Conv1D)            (None, 12, 128)      65536
['SpatialDropout1D_1_3[0][0]']

 SpatialDropout1D_2_3 (SpatialD  (None, 12, 128)     0
['Conv1D_2_3[0][0]']
 ropout1D)

 Conv1D_skipconnection_3 (Conv1  (None, 12, 128)     16512
['residual_Add_2[0][0]']
 D)

 residual_Add_3 (Add)           (None, 12, 128)      0
['SpatialDropout1D_2_3[0][0]',
'Conv1D_skipconnection_3[0][0]']

 lambda_last_timestep (Lambda)  (None, 1)            0
['residual_Add_3[0][0]']

 Dense_singleoutput (Dense)     (None, 1)            2
['lambda_last_timestep[0][0]']

================================================================================
==================
Total params: 512,898
Trainable params: 512,898
Non-trainable params: 0
--------------------------------------------------------------------------------
------------------
```

Tensorboard, enabled by the callback configured in the build, should allow us to observe the NN created.

It has it quirks, and might not run on your machine, if you wish to visualize the NN run `tensorboard --logdir logs/[the date time of the log]`

We check the raw value outputs of the model

[28]:
```
VAL_SIZE = round(len(X) * VAL_SPLIT)

train_data = X[:-VAL_SIZE]
test_data = X[-VAL_SIZE:]
ytrain_data = y[:-VAL_SIZE]
ytest_data = y[-VAL_SIZE:]
print(ytest_data.shape)
print(ytest_data)
```

```
(46,)
[0.39 0.36 0.33 0.45 0.5  0.48 0.43 0.39 0.37 0.39 0.44 0.41 0.4  0.35
 0.32 0.39 0.46 0.48 0.45 0.44 0.35 0.38 0.42 0.43 0.4  0.34 0.32 0.54
```

```
 0.53 0.56 0.48 0.42 0.4  0.43 0.5  0.47 0.46 0.37 0.42 0.54 0.58 0.57
 0.53 0.41 0.41 0.4 ]
```

[29]:
```
y_pred = model.predict(train_data)
yt_pred = model.predict(test_data)

print(yt_pred.shape)
print(yt_pred.flatten())
```

```
6/6 [==============================] - 1s 26ms/step
2/2 [==============================] - 0s 13ms/step
(46, 1)
[0.39172915 0.3632178  0.3683153  0.39611912 0.47268665 0.49147424
 0.4554007  0.40104952 0.39070335 0.40797657 0.42235693 0.41959533
 0.3782336  0.3625284  0.35581696 0.3820904  0.4276475  0.4654602
 0.44974077 0.39825645 0.3829009  0.372997   0.40492934 0.4265596
 0.39837366 0.37735763 0.36668342 0.39269552 0.49389935 0.500273
 0.484464   0.42387706 0.40001166 0.4227609  0.44260973 0.46541047
 0.42005888 0.41167784 0.39291495 0.44419318 0.5177045  0.54010695
 0.50453764 0.4632634  0.42224914 0.44115636]
```

2.3 Loss and Errors

In the graphs below, we visualize how the loss and errors progress through training epochs.

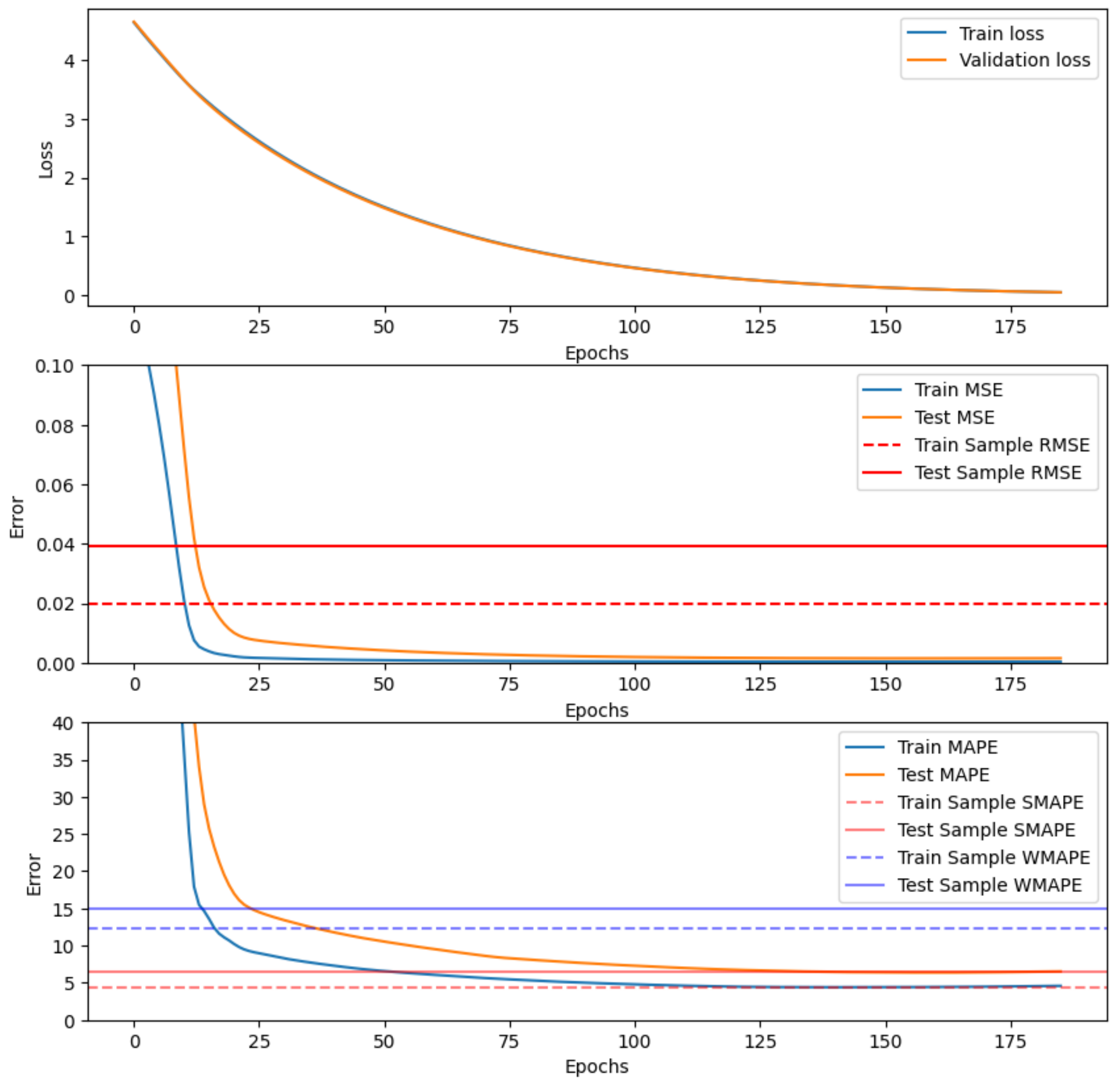
Note how the model might not be generalizing well, given the delta between the training and validation errors.

[30]:
```
from sklearn.metrics import r2_score

rmse_train = mean_squared_error(ytrain_data, y_pred, squared=False)
rmse_test = mean_squared_error(ytest_data, yt_pred, squared=False)
mse_train = mean_squared_error(ytrain_data, y_pred, squared=True)
mse_test = mean_squared_error(ytest_data, yt_pred, squared=True)
smape_train = smape(ytrain_data, y_pred)
smape_test = smape(ytest_data, yt_pred)
wmape_train = wmape(ytrain_data, y_pred)
wmape_test = wmape(ytest_data, yt_pred)


print(f"shapes y_pred: {y_pred.shape} and yt_pred: {yt_pred.shape}")
print(f"RMSE train: {rmse_train:0.04f}")
print(f"RMSE test: {rmse_test:0.04f}")
print(f"MSE train: {mse_train:0.04f}")
print(f"MSE test: {mse_test:0.04f}")
print(f"SMAPE train: {smape_train:0.02f}%")
print(f"SMAPE test: {smape_test:0.02f}%")
print(f"WMAPE train: {wmape_train:0.02f}%")
print(f"WMAPE test: {wmape_test:0.02f}%")
```

```
r2 = r2_score(
    ytest_data,
    yt_pred,
)
print(f"r2: {r2}")

fig, axs = plt.subplots(3, 1, figsize=(10, 10))

axs[0].plot(history.history["loss"], label="Train loss")
axs[0].plot(history.history["val_loss"], label="Validation loss")
axs[0].set_xlabel("Epochs")
axs[0].set_ylabel("Loss")
axs[0].legend()
axs[1].plot(
    history.history["mse"],
    label="Train MSE",
)
axs[1].plot(
    history.history["val_mse"],
    label="Test MSE",
)
axs[1].set_ylim((0, 0.1))
axs[1].axhline(rmse_train, color="r", linestyle="--", label="Train Sample RMSE")
axs[1].axhline(rmse_test, color="r", linestyle="-", label="Test Sample RMSE")
axs[1].set_xlabel("Epochs")
axs[1].set_ylabel("Error")
axs[1].legend()
axs[2].plot(
    history.history["mape"],
    label="Train MAPE",
)
axs[2].plot(
    history.history["val_mape"],
    label="Test MAPE",
)
axs[2].axhline(
    smape_train, color="r", linestyle="--", label="Train Sample SMAPE", alpha=0.
  ↪5
)
axs[2].axhline(
    smape_test, color="r", linestyle="-", label="Test Sample SMAPE", alpha=0.5
)
axs[2].axhline(
    wmape_train, color="b", linestyle="--", label="Train Sample WMAPE", alpha=0.
  ↪5
)
```

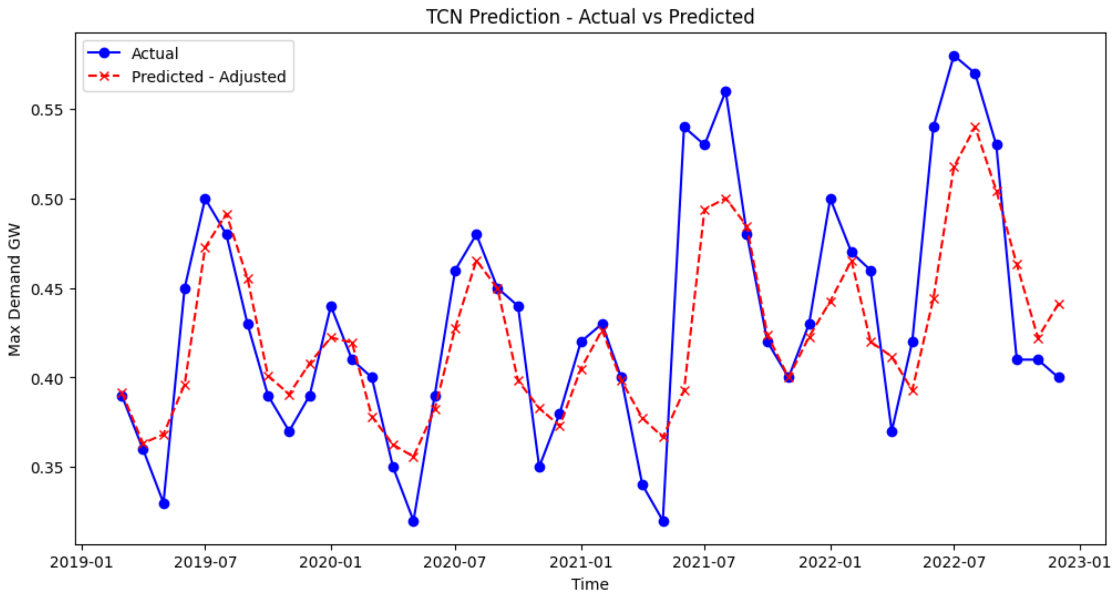
```python
axs[2].axhline(
    wmape_test, color="b", linestyle="-", label="Test Sample WMAPE", alpha=0.5
)
axs[2].set_ylim((0, 40))
axs[2].set_xlabel("Epochs")
axs[2].set_ylabel("Error")
axs[2].legend()
plt.show()
```

```
shapes y_pred: (182, 1) and yt_pred: (46, 1)
RMSE train: 0.0198
RMSE test: 0.0391
MSE train: 0.0004
MSE test: 0.0015
SMAPE train: 4.42%
SMAPE test: 6.50%
WMAPE train: 12.34%
WMAPE test: 14.98%
r2: 0.6578398174795348
```



[31]:
```
plt.figure(figsize=(12, 6))
plt.plot(
    all_data_df.index[-VAL_SIZE:], ytest_data, label="Actual", color="blue",␣
 ↪marker="o"
)
plt.plot(
    all_data_df.index[-VAL_SIZE:],
    yt_pred,
    label="Predicted - Adjusted",
    color="red",
    linestyle="dashed",
    marker="x",
)
```

```python
plt.xlabel("Time")
plt.ylabel("Max Demand GW")
plt.title("TCN Prediction - Actual vs Predicted")
plt.legend()
plt.show()
```



3 Save and Validate Model

We save the model and wieghts.

[32]:
```python
from tensorflow.keras.models import load_model

MODEL_PATH = "./models/tcn"

model.save(MODEL_PATH)
val_model = load_model(MODEL_PATH)

val_model.summary()
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing
5 of 12). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: ./models/tcn\assets

INFO:tensorflow:Assets written to: ./models/tcn\assets
```

```
Model: "TCN"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_4 (InputLayer)           [(None, 12, 7)]      0           []

 Conv1D_1_0 (Conv1D)            (None, 12, 128)      3584
['input_4[0][0]']

 SpatialDropout1D_1_0 (SpatialD  (None, 12, 128)     0
['Conv1D_1_0[0][0]']
 ropout1D)

 Conv1D_2_0 (Conv1D)            (None, 12, 128)      65536
['SpatialDropout1D_1_0[0][0]']

 SpatialDropout1D_2_0 (SpatialD  (None, 12, 128)     0
['Conv1D_2_0[0][0]']
 ropout1D)

 Conv1D_skipconnection_0 (Conv1  (None, 12, 128)     1024
['input_4[0][0]']
 D)

 residual_Add_0 (Add)           (None, 12, 128)      0
['SpatialDropout1D_2_0[0][0]',
'Conv1D_skipconnection_0[0][0]']

 Conv1D_1_1 (Conv1D)            (None, 12, 128)      65536
['residual_Add_0[0][0]']

 SpatialDropout1D_1_1 (SpatialD  (None, 12, 128)     0
['Conv1D_1_1[0][0]']
 ropout1D)

 Conv1D_2_1 (Conv1D)            (None, 12, 128)      65536
['SpatialDropout1D_1_1[0][0]']

 SpatialDropout1D_2_1 (SpatialD  (None, 12, 128)     0
['Conv1D_2_1[0][0]']
 ropout1D)

 Conv1D_skipconnection_1 (Conv1  (None, 12, 128)     16512
['residual_Add_0[0][0]']
 D)
```

```
 residual_Add_1 (Add)           (None, 12, 128)      0
['SpatialDropout1D_2_1[0][0]',
'Conv1D_skipconnection_1[0][0]']

 Conv1D_1_2 (Conv1D)            (None, 12, 128)      65536
['residual_Add_1[0][0]']

 SpatialDropout1D_1_2 (SpatialD  (None, 12, 128)     0
['Conv1D_1_2[0][0]']
 ropout1D)

 Conv1D_2_2 (Conv1D)            (None, 12, 128)      65536
['SpatialDropout1D_1_2[0][0]']

 SpatialDropout1D_2_2 (SpatialD  (None, 12, 128)     0
['Conv1D_2_2[0][0]']
 ropout1D)

 Conv1D_skipconnection_2 (Conv1  (None, 12, 128)     16512
['residual_Add_1[0][0]']
 D)

 residual_Add_2 (Add)           (None, 12, 128)      0
['SpatialDropout1D_2_2[0][0]',
'Conv1D_skipconnection_2[0][0]']

 Conv1D_1_3 (Conv1D)            (None, 12, 128)      65536
['residual_Add_2[0][0]']

 SpatialDropout1D_1_3 (SpatialD  (None, 12, 128)     0
['Conv1D_1_3[0][0]']
 ropout1D)

 Conv1D_2_3 (Conv1D)            (None, 12, 128)      65536
['SpatialDropout1D_1_3[0][0]']

 SpatialDropout1D_2_3 (SpatialD  (None, 12, 128)     0
['Conv1D_2_3[0][0]']
 ropout1D)

 Conv1D_skipconnection_3 (Conv1  (None, 12, 128)     16512
['residual_Add_2[0][0]']
 D)

 residual_Add_3 (Add)           (None, 12, 128)      0
['SpatialDropout1D_2_3[0][0]',
'Conv1D_skipconnection_3[0][0]']
```

```
 lambda_last_timestep (Lambda)  (None, 1)            0
['residual_Add_3[0][0]']

 Dense_singleoutput (Dense)     (None, 1)            2
['lambda_last_timestep[0][0]']

================================================================================
==================
Total params: 512,898
Trainable params: 512,898
Non-trainable params: 0
________________________________________________________________________________
__________________
```

We do some spot predictions.

[33]:
```python
# Our test set has 2yrs, we get the last nov to nov window, and predict dec.
# if this slicing is confusing, we grab the last 13 months (+2) and slice it to␣
↪before the 13th (-1)
window_12month_df = test_df.iloc[-(WINDOW_SIZE_MONTHS + 2) : -1]
ext_test_x, _, _ = prepare_data_and_windows(
    window_12month_df, window=WINDOW_SIZE_MONTHS, horizon=1
)
y_13th_month = val_model.predict(ext_test_x)
print(
    f"For [{test_df.tail(1).index[0]}]: Predicted {y_13th_month[0]} vs Actual␣
↪{test_df.tail(1)[TARGET].values}"
)
```

```
Encoding Widows: 100%|          | 1/1 [00:00<?, ?it/s]

FEATURES: ['Plant_Production_GWh', 'emissions_c02_GG', 'tavg', 'GDP_bln',
'Max_Demand_GW', 'month_sin', 'month_cos'], TARGET: 'Max_Demand_GW', window: 12,
horizon: 1
Shape unencoded (including target label and superflous features): (13, 10)
Shape encoded (window and selected exog features only): (1, 12, 7)
1/1 [==============================] - 0s 300ms/step
For [2022-12-01 00:00:00]: Predicted [0.36517757] vs Actual [0.4]
```