National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Savan Kumar Pandith
Student ID: x22182934

School of Computing
National College of Ireland

Supervisor: Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Savan Kumar Pandith
Student ID:	x22182934
Programme:	Data Analytics **Year:** 2023-2024
Module:	MSc Research Project
Lecturer:	Dr. Catherina Mulwa
Submission Due Date:	……
Project Title:	Cross-Station Solar Power Prediction: A Transfer Learning Approach with Deep learning model
Word Count:	…… **Page Count:**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Savan Kumar Pandith

Date: XXXx

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Savan Kumar Pandith
x22182934

1 Introduction

This report gives the information of all the software and hardware used for this project. Along with the detailed explanation of how all the steps were taken in successfully completing this project.

2 System Configuration

The project hardware information is shown in Fig. 1. This project used 12th Gen Intel processor with the RAM speed of 16GB and 1TB storage.

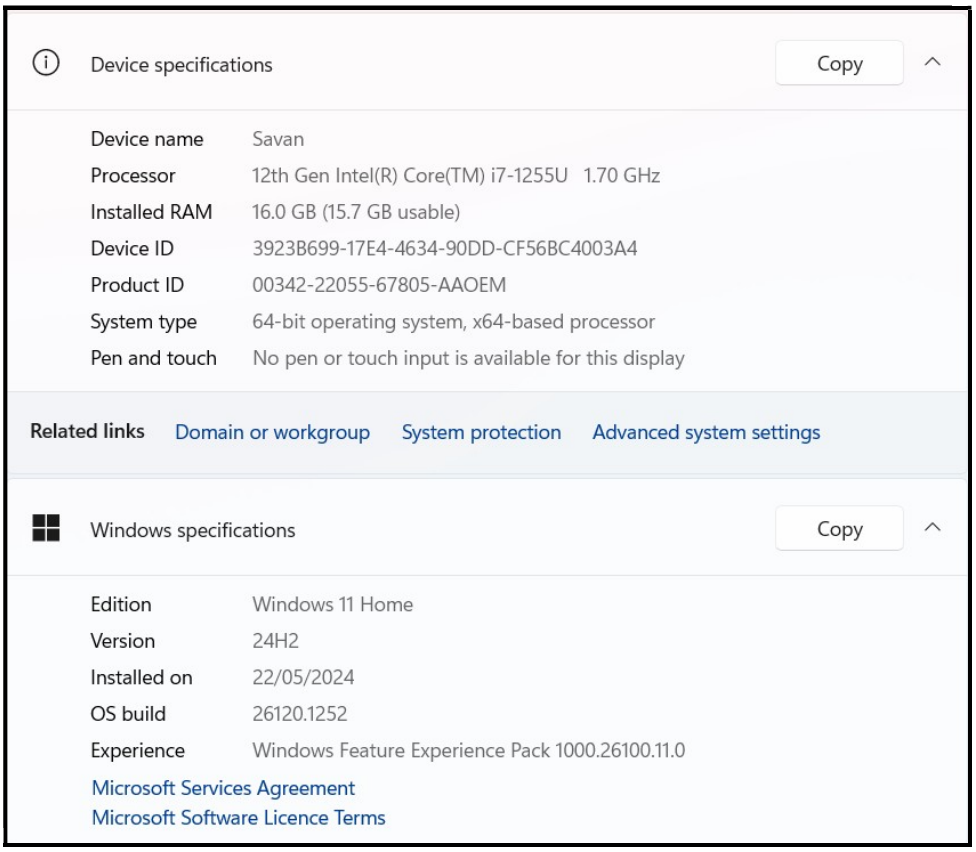

Figure 1: Hardware information

On the software side Python 3 jupyter notebook was used for different tasks and Fig. 2 shows the different libraries used for the implementation of this project.

Importing Libraries

```
import os
import itertools

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow.keras import layers, Sequential
from tensorflow.keras.layers import (
    Conv2D, BatchNormalization, MaxPooling2D, Flatten, Dropout, Dense, LSTM,
    LayerNormalization, Input, Concatenate, Conv1D, Layer
)
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.callbacks import EarlyStopping, Callback

from sklearn.metrics import confusion_matrix, classification_report, mean_squared_error, r2_score, mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestRegressor
from statsmodels.tsa.seasonal import seasonal_decompose
```

Figure. 2: Python libraries

3 Project Implementation

This project is implemented using CRISP-DM methodology method by preprocessing the dataset, building and evaluating the model. Different steps are considered in preparing the model dataset to build and train the model which are discussed below.

3.1 Importing Data

Data was collected under the supervisory control and data acquisition (SCADA) technology for the interval of every 15 minutes and over the period of 2 years from Jan 2019 to Dec 2020. In this research data was first downloaded in .xlsx format from the available repository of the existing study and then it was imported into jupyter notebook by using python libraries pandas as shown in Fig. 3, this library helps to read the excel file and structures into a data frame.

```
import pandas as pd

#Load datasets
Station1_data = pd.read_excel("C:/Users/savan/OneDrive - RampInfoTech Limited/Desktop/Research/solar dataset/Solar station site 1 (Nominal capacity-50MW)
Station2_data = pd.read_excel("C:/Users/savan/OneDrive - RampInfoTech Limited/Desktop/Research/solar dataset/Solar station site 2 (Nominal capacity-130MW
Station3_data = pd.read_excel("C:/Users/savan/OneDrive - RampInfoTech Limited/Desktop/Research/solar dataset/Solar station site 3 (Nominal capacity-30MW)
Station4_data = pd.read_excel("C:/Users/savan/OneDrive - RampInfoTech Limited/Desktop/Research/solar dataset/Solar station site 4 (Nominal capacity-130MW
Station5_data = pd.read_excel("C:/Users/savan/OneDrive - RampInfoTech Limited/Desktop/Research/solar dataset/Solar station site 5 (Nominal capacity-110MW
Station6_data = pd.read_excel("C:/Users/savan/OneDrive - RampInfoTech Limited/Desktop/Research/solar dataset/Solar station site 6 (Nominal capacity-35MW)
Station7_data = pd.read_excel("C:/Users/savan/OneDrive - RampInfoTech Limited/Desktop/Research/solar dataset/Solar station site 7 (Nominal capacity-30MW)
Station8_data = pd.read_excel("C:/Users/savan/OneDrive - RampInfoTech Limited/Desktop/Research/solar dataset/Solar station site 8 (Nominal capacity-30MW)
```

Figure. 3: Import data

The data consisted of 8 solar power stations which ranged from 30 MW to 130 MW and Table.1 shows the nominal capacity of each station along with number of observations for each feature.

	Nominal Capacity	Time(year-month-day h:m:s)	Total solar irradiance (W/m2)	Direct normal irradiance (W/m2)	Global horizontal irradiance (W/m2)	Air temp (°C)	Atmosphere (hpa)	Power (MW)	Relative humidity (%)
Station1	50MW	70176	70176	70176	70176	70176	70176	70176	
Station2	130MW	70176	70176	70176	70176	70176	70176	70176	
Station3	30MW	20352	20352	20352	20352		20352	20352	20352
Station4	130MW	70176	70176	70176	70176	70176	70176	70170	70176
Station5	110MW	70176	70176	70176	70176	70176	70176	70176	70176
Station6	35MW	70176	70176	70176	70176	70176	70176	70176	70176
Station7	30MW	70176	70176	70176	70176	70176	70176	70176	70176
Station8	30MW	69408	69408	69408	69408	69408	69408	69408	69408

Table. 1. Stations & feature count

3.2 Data cleaning

Dataset is then cleaned by checking missing values as shown in Fig. 4, in which we can see 6 missing values in the power variable for solar station4. These missing values are removed by dropping the number of lines as shown in Fig. 5.

```
import pandas as pd

# Function to check for missing values in a dataset
def check_missing_values(data, station_name):
    print(f"--- {station_name} ---")

    missing_values = data.isna().sum()
    print("Missing Values per Column:")
    print(missing_values)

# Checking missing values for each dataset
check_missing_values(Station4_data, "Station 4")
check_missing_values(Station8_data, "Station 8")
```

```
--- Station 4 ---
Missing Values per Column:
Time(year-month-day h:m:s)             0
Total solar irradiance (W/m2)          0
Direct normal irradiance (W/m2)        0
Global horizontal irradiance (W/m2)    0
Air temperature  (°C)                  0
Atmosphere (hpa)                       0
Relative humidity (%)                  0
Power (MW)                             6
dtype: int64
--- Station 8 ---
Missing Values per Column:
Time(year-month-day h:m:s)             0
Total solar irradiance (W/m2)          0
Direct normal irradiance (W/m2)        0
Global horizontal irradiance (W/m2)    0
Air temperature  (°C)                  0
Atmosphere (hpa)                       0
Relative humidity (%)                  0
Power (MW)                             0
dtype: int64
```

Figure. 4: Checking missing values

```
# Removing rows with missing values from Station 4 data
Station4_data = Station4_data.dropna()

# Verifying that rows with missing values have been removed
print("Missing values after cleaning Station 4 data:")
print(Station4_data.isna().sum())
```

```
Missing values after cleaning Station 4 data:
Time(year-month-day h:m:s)             0
Total solar irradiance (W/m2)          0
Direct normal irradiance (W/m2)        0
Global horizontal irradiance (W/m2)    0
Air temperature  (°C)                  0
Atmosphere (hpa)                       0
Relative humidity (%)                  0
Power (MW)                             0
dtype: int64
```

Figure. 5: Dropping missing values

3.3 Feature extraction

As the dataset was in timeseries format. 'Time' column under both stations was converted to a datetime format by using the panda's library. Features like quarter-hour, hour of the day, day of the week, and month were extracted, that helps in understanding temporal aspects like intra-day variations, weekly cycles, and seasonal trends. To improve the model's capacity for identifying temporal relationships in solar power generating data lag features like 'Lag 15min', 'Lag 30min', and 'Lag 1hour' were created by shifting the past values of the 'Power (MW)' variable by 15, 30, and 1 hour as shown in Fig. 6.

```
# Feature extraction
Station4_data['Time'] = pd.to_datetime(Station4_data['Time'])
Station8_data['Time'] = pd.to_datetime(Station8_data['Time'])

# For Station 4
Station4_data['Quarter'] = Station4_data['Time'].dt.minute // 15
Station4_data['Hour'] = Station4_data['Time'].dt.hour
Station4_data['DayOfWeek'] = Station4_data['Time'].dt.dayofweek
Station4_data['Month'] = Station4_data['Time'].dt.month

# For Station 8
Station8_data['Quarter'] = Station8_data['Time'].dt.minute // 15
Station8_data['Hour'] = Station8_data['Time'].dt.hour
Station8_data['DayOfWeek'] = Station8_data['Time'].dt.dayofweek
Station8_data['Month'] = Station8_data['Time'].dt.month

# Adding lag features for Station 4
Station4_data['Lag_15min'] = Station4_data['Power(MW)'].shift(1)
Station4_data['Lag_30min'] = Station4_data['Power(MW)'].shift(2)
Station4_data['Lag_1hour'] = Station4_data['Power(MW)'].shift(4)

# Adding lag features for Station 8
Station8_data['Lag_15min'] = Station8_data['Power(MW)'].shift(1)
Station8_data['Lag_30min'] = Station8_data['Power(MW)'].shift(2)
Station8_data['Lag_1hour'] = Station8_data['Power(MW)'].shift(4)

# Rolling statistics for Station 4
Station4_data['Rolling_Mean_1hour'] = Station4_data['Power(MW)'].rolling(window=4).mean()
Station4_data['Rolling_Std_1hour'] = Station4_data['Power(MW)'].rolling(window=4).std()

# Rolling statistics for Station 8
Station8_data['Rolling_Mean_1hour'] = Station8_data['Power(MW)'].rolling(window=4).mean()
Station8_data['Rolling_Std_1hour'] = Station8_data['Power(MW)'].rolling(window=4).std()
```

Figure. 6: Feature extraction

3.4 Exploratory data analysis

To understand the trend and seasonal variation in the data used seasonal decompose from the python library stats model.tsa.seasonal as show in Fig. 7. Also plotted correlation matrix as shown in Fig. 8 with the help of seaborn and matplotlib library from python

Exploratory data analysis

```
from statsmodels.tsa.seasonal import seasonal_decompose
# Station 4
station4_series = Station4_data_cleaned.set_index('Time')['Power(MW)']
decomposition = seasonal_decompose(station4_series, model='additive', period=96)  # period=96 for daily seasonality (15-min intervals * 24 hours)
decomposition.plot()
plt.show()
# Station 8
station8_series = Station8_data_cleaned.set_index('Time')['Power(MW)']
decomposition = seasonal_decompose(station8_series, model='additive', period=96)
decomposition.plot()
plt.show()
```

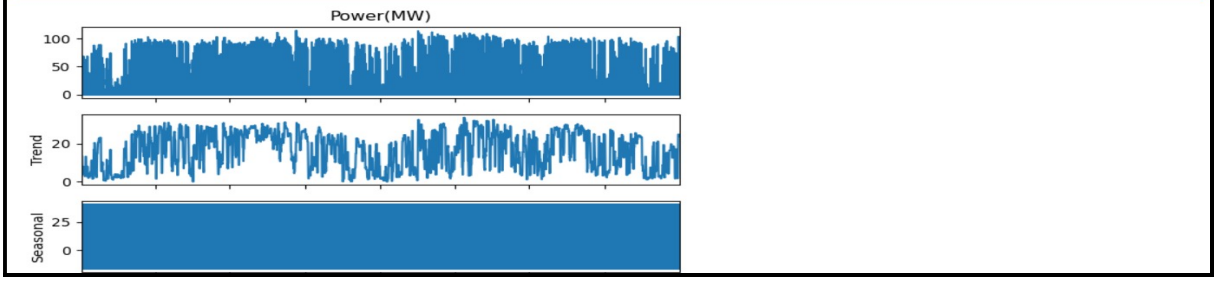

Figure. 7: Season decomposition

```python
import seaborn as sns
import matplotlib.pyplot as plt
def plot_correlation_matrix(dataset, ax, title):
    corr_matrix = dataset.corr()
    sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5, ax=ax)
    ax.set_title(f'Correlation Matrix - {title}')

# Create subplots
fig, axes = plt.subplots(1, 2, figsize=(20, 10))
plot_correlation_matrix(Station4_data_cleaned, axes[0], 'Station 4')
plot_correlation_matrix(Station8_data_cleaned, axes[1], 'Station 8')
plt.tight_layout()
plt.savefig("C:/Users/savan/OneDrive - RampInfoTech Limited/Desktop/Research/solar dataset/correlation_matrices.png")
plt.show()
```

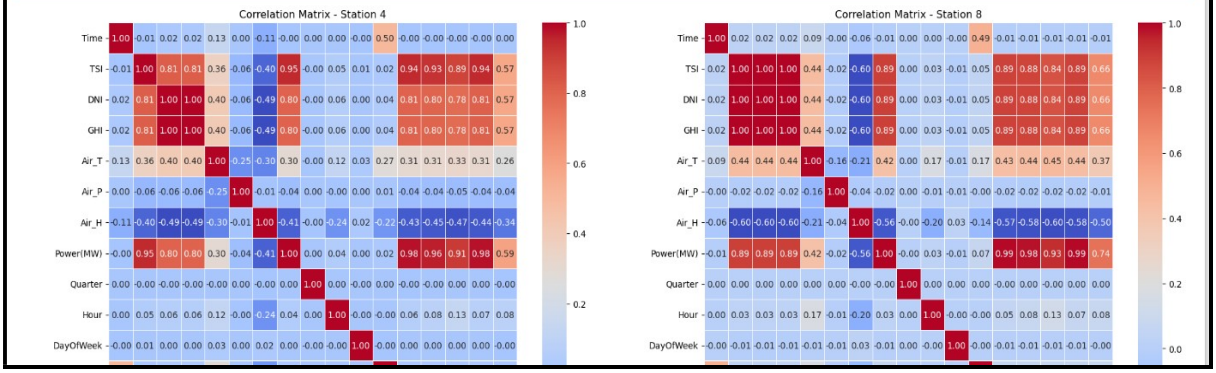

Figure. 8: Correlation matrix

3.5 Feature selection

Feature selection is done with the combination of correlation matrix and random forest regressor technique as shown in Fig. 9. In this process randomforestregressor is used from sklearn python library, which is one of the best techniques to extract important features. Once the features are extracted these are assigned to another variable and reorder and renamed for consistency, Fig. 10 shows the first 5 values of the data set by using head function from pandas librabry.

Feature selection

```python
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
import numpy as np

def exclude_non_numeric(df):
    return df.select_dtypes(include=[np.number])

target_column = 'Power(MW)'

# Station 4
Station4_data_cleaned = exclude_non_numeric(Station4_data_cleaned)
features_4 = Station4_data_cleaned.drop(columns=[target_column])
target_4 = Station4_data_cleaned[target_column]

# Feature selection based on correlation
correlation_matrix_4 = Station4_data_cleaned.corr()
correlation_with_target_4 = correlation_matrix_4[target_column].abs()
selected_features_corr_4 = correlation_with_target_4[correlation_with_target_4 > 0.1].index.tolist()  # Threshold set arbitrarily at 0.1

# Feature importance from Random Forest
rf_4 = RandomForestRegressor(n_estimators=100, random_state=42)
rf_4.fit(features_4, target_4)
importances_4 = rf_4.feature_importances_
indices_4 = np.argsort(importances_4)[::-1]
selected_features_rf_4 = [features_4.columns[i] for i in indices_4 if importances_4[i] > 0.01]  # Threshold set arbitrarily at 0.01
```

Figure. :9 Feature selection using random forest and correlation matrix

```
print(station4_features.head())
print(station8_features.head())


Total Number of Rows in Station 4 Filtered Data:
70166

Total Number of Rows in Station 8 Filtered Data:
69404
    TSI  DNI  GHI  Lag_15min  Lag_30min  Lag_1hour  Rolling_Mean_1hour  \
4   0.0  0.0  0.0     -0.314     -0.314     -0.273             -0.30675
5   0.0  0.0  0.0     -0.314     -0.314     -0.285             -0.31400
6   0.0  0.0  0.0     -0.314     -0.314     -0.314             -0.31400
7   0.0  0.0  0.0     -0.314     -0.314     -0.314             -0.31125
8   0.0  0.0  0.0     -0.303     -0.314     -0.314             -0.31125

    Power(MW)
4      -0.314
5      -0.314
6      -0.314
7      -0.303
8      -0.314
    TSI  DNI  GHI  Lag_15min  Lag_30min  Lag_1hour  Rolling_Mean_1hour  \
4   0.0  0.0  0.0        0.0        0.0        0.0                 0.0
5   0.0  0.0  0.0        0.0        0.0        0.0                 0.0
6   0.0  0.0  0.0        0.0        0.0        0.0                 0.0
7   0.0  0.0  0.0        0.0        0.0        0.0                 0.0
8   0.0  0.0  0.0        0.0        0.0        0.0                 0.0

    Power(MW)
4         0.0
5         0.0
6         0.0
7         0.0
8         0.0
```

Figure. 10: Dataset summary

3.6 Model data preparation

The dataset was first isolated by separating the input features from target variable (Power) for both station 4 and 8. Min-Max scalar function used from sklearn library to normalize both the stations. The scaler fit is applied to the feature and target data of station 8 to ensure consistent scaling as shown in Fig. 11. Also, this step is crucial in this research as the models are trained only on station 8 dataset and then evaluated on station 4, which also relates back to the research question. Next the data for station 8 is split into train (80%) and validation (20%) by using sklearn python library, also complete data on station4 was used for testing as shown in Fig.12.

Modeling- Data preperation

```python
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Separate features and target
X_station4 = station4_features.drop(columns='Power(MW)')
y_station4 = station4_features['Power(MW)']
X_station8 = station8_features.drop(columns='Power(MW)')
y_station8 = station8_features['Power(MW)']

# Min-Max Scaling
scaler_X = MinMaxScaler()
scaler_y = MinMaxScaler()

# Fit scaler on Station 8 data
X_station8_scaled = scaler_X.fit_transform(X_station8)
y_station8_scaled = scaler_y.fit_transform(y_station8.values.reshape(-1, 1))

# Transform Station 4 data using the same scaler
X_station4_scaled = scaler_X.transform(X_station4)
y_station4_scaled = scaler_y.transform(y_station4.values.reshape(-1, 1))
```

Figure. 11: Min max scaling

```python
# Train-Test Split- Station 8 data will be used for training and validation.
X_train = X_station8_scaled
y_train = y_station8_scaled

# Test data from Station 4
X_test = X_station4_scaled
y_test = y_station4_scaled
```

Figure. 12: Train-test split

3.7 Model building

Three hybrid models are built with combination of convolutional neural network (CNN)-long short-term memory (LSTM) and transformer based - multihead attention mechanism by using the library tensorflow and keras as shown in Fig. 13 and Fig. 14.

```python
# Custom Multi-Head Attention Layer
class MultiHeadAttention(Layer):
    def __init__(self, head_count, embed_dim):
        super(MultiHeadAttention, self).__init__()
        self.head_count = head_count
        self.embed_dim = embed_dim
        assert self.embed_dim % self.head_count == 0
        self.depth = self.embed_dim // self.head_count
        self.wq = Dense(self.embed_dim)
        self.wk = Dense(self.embed_dim)
        self.wv = Dense(self.embed_dim)
        self.dense = Dense(self.embed_dim)
    def split_heads(self, x, batch_size):
        x = tf.reshape(x, (batch_size, -1, self.head_count, self.depth))
        return tf.transpose(x, perm=[0, 2, 1, 3])
    def call(self, v, k, q, mask=None):
        batch_size = tf.shape(q)[0]
        q = self.split_heads(self.wq(q), batch_size)
        k = self.split_heads(self.wk(k), batch_size)
        v = self.split_heads(self.wv(v), batch_size)
        scaled_attention, attention_weights = self.scaled_dot_product_attention(q, k, v, mask)
        scaled_attention = tf.transpose(scaled_attention, perm=[0, 2, 1, 3])
        concat_attention = tf.reshape(scaled_attention, (batch_size, -1, self.embed_dim))
        output = self.dense(concat_attention)
        return output
    def scaled_dot_product_attention(self, q, k, v, mask):
        matmul_qk = tf.matmul(q, k, transpose_b=True)
        dk = tf.cast(tf.shape(k)[-1], tf.float32)
        scaled_attention_logits = matmul_qk / tf.math.sqrt(dk)
        if mask is not None:
            scaled_attention_logits += (mask * -1e9)
        attention_weights = tf.nn.softmax(scaled_attention_logits, axis=-1)
        output = tf.matmul(attention_weights, v)
        return output, attention_weights
```

Figure. 13: Transformer model

```python
def build_model(input_shape, head_count=4, embed_dim=64, lstm_units=64):
    inputs = Input(shape=input_shape)
    x = Conv1D(filters=64, kernel_size=3, padding='same', activation='relu')(inputs)
    x = Conv1D(filters=64, kernel_size=3, padding='same', activation='relu')(x)
    x = LSTM(lstm_units, return_sequences=True)(x)
    encoder_output = x

    # Apply multi-head attention
    multi_head_attention = MultiHeadAttention(head_count=head_count, embed_dim=embed_dim)
    attention_output = multi_head_attention(encoder_output, encoder_output, encoder_output, None)

    # Concatenate and apply another attention layer (as in the decoder part)
    concatenated = Concatenate()([encoder_output, attention_output])
    concat_dense = Dense(embed_dim)(concatenated)  # Reduce concatenated dimension
    decoder_attention_output = multi_head_attention(concat_dense, concat_dense, concat_dense, None)

    # Fully connected layer
    output = Dense(1, activation='linear')(decoder_attention_output[:, -1, :])

    Hybrid_Model = Model(inputs=inputs, outputs=output)
    return Hybrid_Model
```

Figure. 14: CNN-LSTM combined

3.8 Model compile and training

All the three hybrid models are trained and validated on station 8 data. For model training, the MSE loss function is used with 50 epochs and an early callback was set to 15 allowing model to train properly and to stop if the validation loss does not improve after 15 epochs as shown in Fig. 15. Libraries used for the model are from keras and sklearn. Similar model building and training procedure is followed for CNN-TF and LSTM- TF model.

```python
# Reshape the data to be compatible with the Conv1D and LSTM layers
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

# Build and compile the model
input_shape = (X_train.shape[1], 1)
Hybrid_Model = build_model(input_shape)
Hybrid_Model.compile(optimizer='adam', loss='mse')

# Early stopping to avoid overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=15, restore_best_weights=True)

# Display the model's summary
Hybrid_Model.summary()

# Train the model on the data
history = Hybrid_Model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2,
                           callbacks=[early_stopping])
```

Figure. 15: Model training

3.9 Model Testing and Evaluation.

All three models are then tested on station 4 dataset and inversed the scaling before calculating the metrics. Metrics are calculated using the sklearn library's, MSE, RMSE, MAE and R2 score as shown in Fig. 16.

```
# Evaluate the model on the test data
y_pred_scaled = Hybrid_Model.predict(X_test)

# Inverse transform the predictions and the true values
y_pred = scaler_y.inverse_transform(y_pred_scaled)
y_test_1 = scaler_y.inverse_transform(y_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test_1, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test_1, y_pred)
r2 = r2_score(y_test_1, y_pred)

print(f'Mean Squared Error (MSE): {mse}')
print(f'Root Mean Squared Error (RMSE): {rmse}')
print(f'Mean Absolute Error (MAE): {mae}')
print(f'R² Score: {r2}')

# Plot predictions vs actual values
plt.figure(figsize=(10, 5))
plt.plot(y_test_1, label='Actual Values')
plt.plot(y_pred, label='Predicted Values')
plt.title('Actual vs Predicted Power (MW)')
plt.xlabel('Sample')
plt.ylabel('Power (MW)')
plt.legend()
plt.show()
```

Figure. 16: Model test & evaluation