Career Foundry Data Analytics Program

Machine Learning Specialization

Luis Gil

Task 2.2 Report: Data Cleaning and RNN Model Development

Introduction

In this task, we carried out two key activities: data cleaning and preparation for a time-series classification problem and developing a Recurrent Neural Network (RNN) model to classify the weather conditions as pleasant or not for different cities. The steps involved in this task are documented across two main scripts: one for data wrangling and cleaning, and another for developing the machine learning model.

Data Cleaning and Wrangling

The dataset consisted of 170 columns representing different weather variables for various cities. The task focused on cleaning and preparing the data for machine learning.

1. **Initial Dataset Overview**: The dataset contained 22,950 entries with weather observations for several European cities. We used columns such as temp_mean, wind_speed, humidity, and pressure, among others, as features. However, some columns related to cities like Gdansk, Roma, and Tours were not needed for model training.

In [3]:
```
# Import the orders data
weather_data = pd.read_csv(os.path.join(path, 'Data Sets', 'Weather.csv'))
weather_data
```

Out[3]:

	DATE	MONTH	BASEL_cloud_cover	BASEL_wind_speed	BASEL_humidity	BASEL_pressure	BASEL_global_radiation	BASEL_precipitation	BASEL_s
0	19600101	1	7	2.1	0.85	1.0180	0.32	0.09	
1	19600102	1	6	2.1	0.84	1.0180	0.36	1.05	
2	19600103	1	8	2.1	0.90	1.0180	0.18	0.30	
3	19600104	1	3	2.1	0.92	1.0180	0.58	0.00	
4	19600105	1	6	2.1	0.95	1.0180	0.65	0.14	
...	...	...	...	...	...	...	...	...	
22945	20221027	10	1	2.1	0.79	1.0248	1.34	0.22	
22946	20221028	10	6	2.1	0.77	1.0244	1.34	0.22	
22947	20221029	10	4	2.1	0.76	1.0227	1.34	0.22	
22948	20221030	10	5	2.1	0.80	1.0212	1.34	0.22	
22949	20221031	10	5	2.1	0.84	1.0193	1.34	0.22	

22950 rows × 170 columns

2. **Column Removal**: We removed all columns related to Gdansk, Roma, and Tours, as they were not included in the pleasant weather dataset. Additionally, irrelevant observations such as wind_speed and snow_depth were dropped from the weather data since they were not essential for the model.

In [8]:
```
# List of cities to be removed
cities_to_remove = ['GDANSK', 'ROMA', 'TOURS']

# Filter out columns that contain the names of the cities to remove
columns_to_keep = [col for col in weather_data.columns if not any(city in col for city in cities_to_remove)]

# Create a new dataframe with the filtered columns
weather_data_filtered = weather_data[columns_to_keep]

# Check the shape to ensure the columns were removed
print(f"Shape of weather_data_filtered: {weather_data_filtered.shape}")
```

```
Shape of weather_data_filtered: (22950, 149)
```

3. **Column Addition**: To ensure we had sufficient data for model training, three new columns (cloud_cover, pressure, and humidity) were added for three locations (Kassel, Sonnblick, and Oslo). We used data from nearby stations: Kassel data was copied from Ljubljana, Sonnblick from Munchen, and Oslo from Stockholm.

In [10]:
```python
# Add new columns to weather_data_cleaned and copy values from nearby stations

# Copy cloud_cover, pressure, and humidity from Ljubljana to Kassel
weather_data_cleaned['KASSEL_cloud_cover'] = weather_data_cleaned['LJUBLJANA_cloud_cover']
weather_data_cleaned['KASSEL_pressure'] = weather_data_cleaned['LJUBLJANA_pressure']
weather_data_cleaned['KASSEL_humidity'] = weather_data_cleaned['LJUBLJANA_humidity']

# Copy cloud_cover, pressure, and humidity from Munchen to Sonnblick
weather_data_cleaned['SONNBLICK_cloud_cover'] = weather_data_cleaned['MUNCHENB_cloud_cover']
weather_data_cleaned['MUNCHENB_pressure'] = weather_data_cleaned['SONNBLICK_pressure']
weather_data_cleaned['SONNBLICK_humidity'] = weather_data_cleaned['MUNCHENB_humidity']

# Copy cloud_cover, pressure, and humidity from Stockholm to Oslo
weather_data_cleaned['OSLO_cloud_cover'] = weather_data_cleaned['STOCKHOLM_cloud_cover']
weather_data_cleaned['OSLO_pressure'] = weather_data_cleaned['STOCKHOLM_pressure']
weather_data_cleaned['STOCKHOLM_humidity'] = weather_data_cleaned['OSLO_humidity']

# Confirm that the columns have been added successfully
print(weather_data_cleaned[['KASSEL_cloud_cover', 'SONNBLICK_cloud_cover', 'OSLO_cloud_cover']].head())
print(weather_data_cleaned[['KASSEL_pressure', 'SONNBLICK_pressure', 'OSLO_pressure']].head())
print(weather_data_cleaned[['KASSEL_humidity', 'SONNBLICK_humidity', 'OSLO_humidity']].head())
```

```
   KASSEL_cloud_cover  SONNBLICK_cloud_cover  OSLO_cloud_cover
0                   8                      5                 5
1                   6                      6                 5
2                   8                      6                 5
3                   6                      6                 5
4                   7                      5                 5
```

4. **Final Dataset**: After cleaning, the dataset contained 134 relevant features across 22,950 samples. The data was reshaped for the neural network, maintaining a final shape of (22,950, 15, 9) for the feature set (X), which was split into training and test sets for model development.

In [12]:
```python
# Drop DATE and MONTH from weather data
weather_data_cleaned = weather_data_cleaned.drop(columns=['DATE', 'MONTH'])
weather_data_cleaned
```

Out[12]:

	BASEL_cloud_cover	BASEL_humidity	BASEL_pressure	BASEL_global_radiation	BASEL_precipitation	BASEL_sunshine	BASEL_temp_mean	BASEL_tem
0	7	0.85	1.0180	0.32	0.09	0.7	6.5	
1	6	0.84	1.0180	0.36	1.05	1.1	6.1	
2	8	0.90	1.0180	0.18	0.30	0.0	8.5	
3	3	0.92	1.0180	0.58	0.00	4.1	6.3	
4	6	0.95	1.0180	0.65	0.14	5.4	3.0	
...	...	...	...	...	...	...	...	
22945	1	0.79	1.0248	1.34	0.22	7.7	15.9	
22946	6	0.77	1.0244	1.34	0.22	5.4	16.7	
22947	4	0.76	1.0227	1.34	0.22	6.1	16.7	
22948	5	0.80	1.0212	1.34	0.22	5.8	15.4	
22949	5	0.84	1.0193	1.34	0.22	3.2	13.5	

22950 rows × 135 columns

In [9]:
```python
# Assuming X is 2D array or DataFrame with shape (22950*15, 9)

X = X.reshape(-1, 15, 9)
print("Reshaped X:", X.shape)  # Should output (22950, 15, 9)
```

```
Reshaped X: (22950, 15, 9)
```

Data Preparation

The dataset was split into training and test sets, with 80% of the data used for training and 20% for testing. The final shapes of the training and testing sets were:

- X_train: (18,360, 15, 9)
- y_train: (18,360, 15)
- X_test: (4,590, 15, 9)
- y_test: (4,590, 15)

```
In [10]: #Split data into training and testing sets.
         X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)

In [11]: print(X_train.shape, y_train.shape)
         print(X_test.shape, y_test.shape)

         (18360, 15, 9) (18360, 15)
         (4590, 15, 9) (4590, 15)

In [12]: X_train

Out[12]: array([[[ 1.0000e+00,  6.0000e-01,  1.0180e+00, ...,  1.6400e+01,
                   7.3000e+00,  2.3100e+01],
                 [ 6.0000e+00,  6.4000e-01,  1.0146e+00, ...,  2.3200e+01,
                   2.1000e+01,  2.8900e+01],
                 [ 4.0000e+00,  6.7000e-01,  1.0170e+00, ...,  2.1100e+01,
                   1.7400e+01,  2.6700e+01],
                 ...,
                 [ 1.0302e+00,  1.5300e+00,  1.7300e+00, ...,  3.4000e+00,
                   8.0000e+00,  1.0106e+00],
                 [ 6.9000e-01,  6.9000e-01,  0.0000e+00, ...,  7.0000e+00,
                   7.8000e-01,  1.0229e+00],
                 [ 2.3800e+00,  0.0000e+00,  5.3000e+00, ...,  6.0000e+00,
                   1.0302e+00,  8.7000e-01]]
```

Model Architecture

We created multiple **Recurrent Neural Network (RNN)** models using Keras for weather prediction at 15 different European stations. RNN models, particularly LSTMs (Long Short-Term Memory), are designed to handle temporal sequence data, making them suitable for weather time series analysis.

Key Models Tested:

Model Report 1: CNN Model with Activation Type tanh, 30 Epochs, 32 Batch Size, 128 Hidden Units

Model Architecture:

- **Convolution Layer**: Conv1D with 128 filters, kernel size 2, and activation relu.
- **Dense Layer**: 16 units with activation relu.
- **Pooling Layer**: MaxPooling1D to reduce dimensionality.
- **Flatten Layer**: Converts the 3D tensor to 1D for further processing.
- **Output Layer**: Dense layer with the tanh activation for multi-class classification.
- **Loss Function**: Categorical Crossentropy.
- **Optimizer**: Adam.

```
In [13]: # Create a Keras Leyered model. Change activation type : 30, 32, 128, tanh
epochs = 30
batch_size = 32
n_hidden = 128

timesteps = len(X_train[0])
input_dim = len(X_train[0][0])
n_classes = len(y_train[0])

model = Sequential()
model.add(Conv1D(n_hidden, kernel_size=2, activation='relu', input_shape=(timesteps, input_dim)))
model.add(Dense(16, activation='relu'))
model.add(MaxPooling1D())
model.add(Flatten())
model.add(Dense(n_classes, activation='tanh'))

C:\Users\luis\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an `input_s
hape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer i
n the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

In [14]: model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Results:

Training Process (Epochs 1-30):

- **Initial Loss (Epoch 1)**: 23.30
- **Initial Accuracy (Epoch 1)**: 4.06%
- **Validation Accuracy (Epoch 1)**: 3.94%
- **Final Loss (Epoch 30)**: 22.57
- **Final Accuracy (Epoch 30)**: 9.04%
- **Validation Accuracy (Epoch 30)**: 9.52%

The accuracy improved slightly from 4.06% to 9.04%, indicating slow learning. However, the loss remained high throughout the epochs, suggesting the model struggled to effectively classify the data.

Confusion Matrix:

In [18]:
```
# Evaluate
print(confusion_matrix(y_test, model.predict(X_test)))
```

```
144/144 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
Pred        BASEL  BUDAPEST  MAASTRICHT  MUNCHENB
True
BASEL         270      2592          26        21
BELGRADE      220       681           7         1
BUDAPEST       14       167           0         0
DEBILT         12        59           0         0
DUSSELDORF      1        25           0         0
HEATHROW        3        83           0         0
KASSEL          0         6           0         0
LJUBLJANA       1        48           0         0
MAASTRICHT      0         4           0         0
MADRID          8       323           0         0
MUNCHENB        1         5           0         0
OSLO            0         7           0         0
STOCKHOLM       0         2           0         0
VALENTIA        0         3           0         0
```

Observations:

- **BASEL** was recognized correctly only 270 times, with most predictions (2,592) being incorrect, falling under **BUDAPEST**.
- **BUDAPEST** itself had 167 correct predictions but many other labels were incorrectly classified as BUDAPEST.
- Other stations like **DEBILT**, **DUSSELDORF**, and **HEATHROW** had very low or no correct classifications, indicating that the model struggles to differentiate stations effectively.

Conclusions for model 1:

- **Low Accuracy**: The accuracy remained low (under 10%) and the model showed little improvement during training.
- **High Confusion**: The confusion matrix shows that the model often predicted the wrong station, particularly favoring BUDAPEST.
- **Slow Learning**: Despite multiple epochs, the loss did not decrease significantly and the model's ability to generalize remained weak.

Luis Gil

Model Report 2: CNN Model with activation type sigmoid, using 20 epochs and 128 LSTM units

- **Layer 1:** LSTM with 128 units, using sigmoid activation function, and a Dropout rate of 0.5.
- **Layer 2:** Dense layer with output size equal to the number of classes, using sigmoid activation for multi-class classification.
- **Optimizer:** RMSprop
- **Loss Function:** categorical_crossentropy
- **Metrics:** Accuracy

```
In [19]: # Create a Keras leyered model. Change activation type: 20, 32, 128, sigmoid
epochs = 20
batch_size = 32
n_hidden = 128

timesteps = len(X_train[0])
input_dim = len(X_train[0][0])
n_classes = len(y_train[0])

model = Sequential()
model.add(LSTM(n_hidden, input_shape=(timesteps, input_dim)))
model.add(Dropout(0.5))
model.add(Dense(n_classes, activation='sigmoid'))
```

Training and Validation Results:

- **Epoch 1:**
  - Training Accuracy: 8.15%
  - Validation Accuracy: 6.41%
  - Loss: 10.95 (train), 9.39 (validation)
- **Epoch 20:**
  - Training Accuracy: 7.32%
  - Validation Accuracy: 3.70%
  - Loss: 17.24 (train), 16.65 (validation)

The model's accuracy remained relatively low throughout training, with validation accuracy decreasing slightly to 3.70%. The loss started at 10.95 and increased to 17.24, indicating that the model did not converge well on the data. This might suggest that the model struggled to differentiate between the stations or that the sigmoid activation function and other hyperparameters were not ideal for this task.

```
In [21]: model.fit(X_train,
                   y_train,
                   batch_size=batch_size,
                   validation_data=(X_test, y_test),
                   epochs=epochs)
```

```
Epoch 1/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 20s 21ms/step - accuracy: 0.0815 - loss: 10.9481 - val_accuracy: 0.0641 - val_loss: 9.3890
Epoch 2/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 19s 18ms/step - accuracy: 0.0856 - loss: 11.7965 - val_accuracy: 0.0403 - val_loss: 9.9814
Epoch 3/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 21s 19ms/step - accuracy: 0.0792 - loss: 12.1299 - val_accuracy: 0.0377 - val_loss: 10.2928
Epoch 4/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 20s 18ms/step - accuracy: 0.0812 - loss: 12.5784 - val_accuracy: 0.0407 - val_loss: 10.8550
Epoch 5/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 21s 19ms/step - accuracy: 0.0854 - loss: 13.0747 - val_accuracy: 0.0501 - val_loss: 11.2788
Epoch 6/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 21s 19ms/step - accuracy: 0.0827 - loss: 12.9847 - val_accuracy: 0.0503 - val_loss: 11.6400
Epoch 7/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - accuracy: 0.0831 - loss: 13.5652 - val_accuracy: 0.0447 - val_loss: 12.0741
Epoch 8/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 11s 20ms/step - accuracy: 0.0792 - loss: 13.9235 - val_accuracy: 0.0490 - val_loss: 12.4594
Epoch 9/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 21s 20ms/step - accuracy: 0.0841 - loss: 14.2430 - val_accuracy: 0.0586 - val_loss: 12.8651
Epoch 10/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 20s 19ms/step - accuracy: 0.0839 - loss: 14.4797 - val_accuracy: 0.0499 - val_loss: 13.2113
Epoch 11/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 20s 19ms/step - accuracy: 0.0858 - loss: 14.8152 - val_accuracy: 0.0383 - val_loss: 13.5398
Epoch 12/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 11s 18ms/step - accuracy: 0.0864 - loss: 14.9788 - val_accuracy: 0.0261 - val_loss: 14.0155
Epoch 13/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 11s 18ms/step - accuracy: 0.0846 - loss: 15.2109 - val_accuracy: 0.0595 - val_loss: 14.3910
Epoch 14/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 21s 19ms/step - accuracy: 0.0860 - loss: 15.5934 - val_accuracy: 0.0305 - val_loss: 14.7281
Epoch 15/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 20s 19ms/step - accuracy: 0.0776 - loss: 15.9998 - val_accuracy: 0.0279 - val_loss: 15.0832
Epoch 16/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 11s 19ms/step - accuracy: 0.0827 - loss: 16.3812 - val_accuracy: 0.0519 - val_loss: 15.2604
Epoch 17/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 21s 19ms/step - accuracy: 0.0793 - loss: 15.9578 - val_accuracy: 0.0394 - val_loss: 15.6662
Epoch 18/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 20s 19ms/step - accuracy: 0.0792 - loss: 16.6846 - val_accuracy: 0.0377 - val_loss: 15.7617
Epoch 19/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 21s 19ms/step - accuracy: 0.0808 - loss: 16.7077 - val_accuracy: 0.0503 - val_loss: 16.3914
Epoch 20/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 21s 20ms/step - accuracy: 0.0732 - loss: 17.2414 - val_accuracy: 0.0370 - val_loss: 16.6460

Out[21]: <keras.src.callbacks.history.History at 0x1afb4d559d0>
```

Confusion Matrix (Predictions):

The confusion matrix reveals that the model heavily predicted the **BASEL** station for most classes. For example:

- **BASEL** was correctly classified 2,909 times.
- **BELGRADE**, **BUDAPEST**, **DEBILT**, etc., were almost entirely classified as **BASEL**.
- This indicates a significant imbalance in predictions, where the model tends to predict one class (BASEL) for most of the test data.

Luis Gil

```
In [22]: # Evaluate
         print(confusion_matrix(y_test, model.predict(X_test)))

144/144 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step
Pred         BASEL
True
BASEL         2909
BELGRADE       909
BUDAPEST       181
DEBILT          71
DUSSELDORF      26
HEATHROW        86
KASSEL           6
LJUBLJANA       49
MAASTRICHT       4
MADRID         331
MUNCHENB         6
OSLO             7
STOCKHOLM        2
VALENTIA         3
```

Analysis:

- **Performance:** The model's poor performance, with validation accuracy dropping to 3.70%, suggests that the sigmoid activation function and current LSTM structure may not be suitable for this problem.

- **Possible Issues:**
  - The use of sigmoid for multi-class classification could be contributing to the poor performance, as sigmoid is generally used for binary classification. A better choice might be the **softmax** activation function, which is more commonly used for multi-class classification.
  - The data might also need further preprocessing or normalization to help the model better differentiate between different weather stations.
  - Additionally, increasing the model complexity or adding more convolutional layers could improve feature extraction.

Model Report 3: CNN Model with activation type tanh, using 20 epochs and 128 LSTM units

- **Layer 1:** LSTM with 128 units, using tanh activation function, and a Dropout rate of 0.5.
- **Layer 2:** Dense layer with output size equal to the number of classes (15), using tanh activation.
- **Optimizer:** RMSprop
- **Loss Function:** categorical_crossentropy
- **Metrics:** Accuracy

```
In [23]: # Create a Keras Leyered model. Change activation type: 20, 32, 128, tanh
epochs = 20
batch_size = 32
n_hidden = 128

timesteps = len(X_train[0])
input_dim = len(X_train[0][0])
n_classes = len(y_train[0])

model = Sequential()
model.add(LSTM(n_hidden, input_shape=(timesteps, input_dim)))
model.add(Dropout(0.5))
model.add(Dense(n_classes, activation='tanh'))
```

Training and Validation Results:

- **Epoch 1:**
  - Training Accuracy: 1.85%
  - Validation Accuracy: 3.94%
  - Loss: 24.26 (train), 25.28 (validation)
- **Epoch 20:**
  - Training Accuracy: 1.79%
  - Validation Accuracy: 0.15%
  - Loss: 24.76 (train), 17.43 (validation)

```
Epoch 1/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 20s 22ms/step - accuracy: 0.0185 - loss: 24.2621 - val_accuracy: 0.0394 - val_loss: 25.2865
Epoch 2/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 19s 19ms/step - accuracy: 0.0411 - loss: 25.1591 - val_accuracy: 0.0394 - val_loss: 25.4206
Epoch 3/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 21s 20ms/step - accuracy: 0.0356 - loss: 25.1860 - val_accuracy: 0.0394 - val_loss: 29.7667
Epoch 4/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 20s 19ms/step - accuracy: 0.0296 - loss: 25.0249 - val_accuracy: 0.0379 - val_loss: 24.9751
Epoch 5/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - accuracy: 0.0284 - loss: 25.1532 - val_accuracy: 0.0394 - val_loss: 25.8419
Epoch 6/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 11s 18ms/step - accuracy: 0.0325 - loss: 24.6446 - val_accuracy: 0.0394 - val_loss: 29.2309
Epoch 7/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 21s 18ms/step - accuracy: 0.0278 - loss: 24.5465 - val_accuracy: 0.0394 - val_loss: 24.1974
Epoch 8/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - accuracy: 0.0296 - loss: 24.7787 - val_accuracy: 0.0394 - val_loss: 20.2635
Epoch 9/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - accuracy: 0.0311 - loss: 24.9145 - val_accuracy: 0.0394 - val_loss: 28.1900
Epoch 10/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - accuracy: 0.0353 - loss: 24.8523 - val_accuracy: 0.0394 - val_loss: 23.9435
Epoch 11/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - accuracy: 0.0337 - loss: 25.2041 - val_accuracy: 0.0394 - val_loss: 26.4676
Epoch 12/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 10s 17ms/step - accuracy: 0.0349 - loss: 24.8042 - val_accuracy: 0.0394 - val_loss: 27.5275
Epoch 13/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - accuracy: 0.0350 - loss: 24.5558 - val_accuracy: 0.0394 - val_loss: 22.9134
Epoch 14/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 11s 19ms/step - accuracy: 0.0301 - loss: 24.2461 - val_accuracy: 0.0394 - val_loss: 24.0704
Epoch 15/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - accuracy: 0.0340 - loss: 24.8812 - val_accuracy: 0.0394 - val_loss: 25.8069
Epoch 16/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - accuracy: 0.0326 - loss: 24.3950 - val_accuracy: 0.0394 - val_loss: 28.6937
Epoch 17/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - accuracy: 0.0308 - loss: 24.6882 - val_accuracy: 0.0394 - val_loss: 23.6712
Epoch 18/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - accuracy: 0.0311 - loss: 24.4688 - val_accuracy: 0.0390 - val_loss: 23.2135
Epoch 19/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - accuracy: 0.0299 - loss: 24.8244 - val_accuracy: 0.0390 - val_loss: 24.2801
Epoch 20/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 11s 18ms/step - accuracy: 0.0179 - loss: 24.7624 - val_accuracy: 0.0015 - val_loss: 17.4328

<keras.src.callbacks.history.History at 0x1afb6e82910>
```

Analysis:

- **Performance:** This model's accuracy was notably low, and it struggled to converge. The training accuracy started at 1.85% and decreased to 1.79% by the 20th epoch. The validation accuracy began at 3.94% but dropped to just **0.15%** by the last epoch, showing that the model didn't generalize well.

- **Loss:** The loss values remained high throughout the training process, with only minor improvements. The training loss started at 24.26 and ended at 24.76, while the validation loss improved slightly but remained high (from 25.28 to 17.43).

These values indicate that the model was unable to effectively differentiate between the different weather stations or learn the relationships within the dataset.

Confusion Matrix (Predictions):

The confusion matrix reveals that the model heavily predicted the **OSLO** station for almost all the test data. For example:

- **BASEL** was predicted as **OSLO** 2,908 times and as **MAASTRICHT** once.
- All other true classes were predicted as **OSLO** as well, suggesting that the model became highly biased toward one station (OSLO), leading to an almost complete misclassification of the test data.

```
In [26]: # Evaluate
         print(confusion_matrix(y_test, model.predict(X_test)))

144/144 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step
Pred        MAASTRICHT  OSLO
True
BASEL                1  2908
BELGRADE             0   909
BUDAPEST             0   181
DEBILT               0    71
DUSSELDORF           0    26
HEATHROW             0    86
KASSEL               0     6
LJUBLJANA            0    49
MAASTRICHT           0     4
MADRID               0   331
MUNCHENB             0     6
OSLO                 0     7
STOCKHOLM            0     2
VALENTIA             0     3
```

Summary:

- **Performance:** The model did not perform well, achieving very low training and validation accuracies, with loss values remaining high. The **tanh** activation function may not have been suitable in this context, or the model may require more tuning (such as the addition of more layers, changing the optimizer, or adjusting the number of units).
- **Confusion Matrix:** The predictions were dominated by the **OSLO** station, showing that the model failed to differentiate between the various weather stations.

Luis Gil

Model Report 4: CNN Model with activation type sigmoid, using 20 epochs and Conv1D, MaxPooling1D, and LSTM layers.

- **Layer 1:** 1D Convolutional layer with 64 filters, a kernel size of 3, and relu activation.
- **Layer 2:** MaxPooling1D layer with a pool size of 2.
- **Layer 3:** 1D Convolutional layer with 128 filters, a kernel size of 3, and relu activation.
- **Layer 4:** MaxPooling1D layer with a pool size of 2.
- **Layer 5:** LSTM layer with 128 units.
- **Layer 6:** Dropout layer with a dropout rate of 0.5.
- **Layer 7:** Dense layer with output size equal to the number of classes (15), using sigmoid activation.
- **Optimizer:** RMSprop
- **Loss Function:** categorical_crossentropy
- **Metrics:** Accuracy

In [27]:
```
# Create a Keras Leyered model. Hyperparameters: 20, 32, 128, sigmoid. Add Convolution and Pooling Layers
epochs = 20
batch_size = 32
n_hidden = 128

timesteps = len(X_train[0])
input_dim = len(X_train[0][0])
n_classes = len(y_train[0])

model = Sequential()
model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(timesteps, input_dim)))
model.add(MaxPooling1D(pool_size=2))
model.add(Conv1D(filters=128, kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(LSTM(n_hidden))
model.add(Dropout(0.5))
model.add(Dense(n_classes, activation='sigmoid'))
```

Training and Validation Results:

- **Epoch 1:**
  - Training Accuracy: 8.88%
  - Validation Accuracy: 7.47%
  - Loss: 10.77 (train), 10.09 (validation)
- **Epoch 20:**
  - Training Accuracy: 7.53%
  - Validation Accuracy: 7.43%
  - Loss: 27.69 (train), 27.13 (validation)

Luis Gil

```
Epoch 1/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 18s 14ms/step - accuracy: 0.0888 - loss: 10.7693 - val_accuracy: 0.0747 - val_loss: 10.0924
Epoch 2/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 6s 11ms/step - accuracy: 0.0985 - loss: 11.9949 - val_accuracy: 0.0747 - val_loss: 11.3062
Epoch 3/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 6s 11ms/step - accuracy: 0.1012 - loss: 13.0364 - val_accuracy: 0.0756 - val_loss: 12.4324
Epoch 4/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - accuracy: 0.0975 - loss: 14.1951 - val_accuracy: 0.0739 - val_loss: 13.3811
Epoch 5/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 7s 13ms/step - accuracy: 0.0898 - loss: 14.8378 - val_accuracy: 0.0728 - val_loss: 14.2002
Epoch 6/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 9s 11ms/step - accuracy: 0.0970 - loss: 15.8786 - val_accuracy: 0.0741 - val_loss: 15.0166
Epoch 7/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 6s 11ms/step - accuracy: 0.0899 - loss: 16.4437 - val_accuracy: 0.0736 - val_loss: 15.9706
Epoch 8/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - accuracy: 0.0968 - loss: 17.4057 - val_accuracy: 0.0765 - val_loss: 17.1395
Epoch 9/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - accuracy: 0.0981 - loss: 18.4748 - val_accuracy: 0.0743 - val_loss: 18.0504
Epoch 10/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 6s 11ms/step - accuracy: 0.0936 - loss: 18.9375 - val_accuracy: 0.0732 - val_loss: 18.9789
Epoch 11/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - accuracy: 0.0893 - loss: 20.0912 - val_accuracy: 0.0730 - val_loss: 19.7796
Epoch 12/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - accuracy: 0.0906 - loss: 20.9480 - val_accuracy: 0.0721 - val_loss: 20.7878
Epoch 13/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - accuracy: 0.0930 - loss: 21.9100 - val_accuracy: 0.0743 - val_loss: 21.7743
Epoch 14/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - accuracy: 0.0861 - loss: 22.2770 - val_accuracy: 0.0732 - val_loss: 22.3260
Epoch 15/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - accuracy: 0.0828 - loss: 24.0202 - val_accuracy: 0.0739 - val_loss: 23.3617
Epoch 16/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - accuracy: 0.0795 - loss: 24.5565 - val_accuracy: 0.0723 - val_loss: 24.2363
Epoch 17/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - accuracy: 0.0812 - loss: 25.0408 - val_accuracy: 0.0721 - val_loss: 24.7923
Epoch 18/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 7s 12ms/step - accuracy: 0.0838 - loss: 25.8417 - val_accuracy: 0.0732 - val_loss: 25.8532
Epoch 19/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - accuracy: 0.0758 - loss: 26.0189 - val_accuracy: 0.0730 - val_loss: 26.4636
Epoch 20/20
574/574 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - accuracy: 0.0753 - loss: 27.6935 - val_accuracy: 0.0743 - val_loss: 27.1276
```

Analysis:

- **Performance:** The model started with reasonable accuracy of around 8.88% but did not improve significantly as the epochs progressed. The training accuracy dropped slightly to 7.53%, and the validation accuracy remained very close at 7.43%.
- **Loss:** The loss remained high and even increased as the training continued, going from 10.77 to 27.69 in training and from 10.09 to 27.13 in validation. This suggests that the model was overfitting or unable to generalize well to the validation data.
- **Sigmoid Activation:** The use of sigmoid activation in the final layer may have limited the model's performance in this multi-class classification problem. Sigmoid is typically used for binary classification, whereas softmax might have been a more appropriate choice for multi-class scenarios.

Confusion Matrix (Predictions):

The confusion matrix shows that the model predominantly predicted the **BASEL** station:

- **BASEL:** Correctly predicted **2,909** times.
- **BELGRADE:** Only correctly predicted **once**.
- **MADRID:** Correctly predicted **zero** times.

The model was heavily biased towards **BASEL**, predicting this station for almost all cases, leading to poor generalization for the other stations. Most of the true classes, including **BUDAPEST**, **DEBILT**, **DUSSELDORF**, **HEATHROW**, and others, were all misclassified as **BASEL**.

```
144/144 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step
Pred         BASEL  BELGRADE  MADRID
True
BASEL         2909         0       0
BELGRADE       908         1       0
BUDAPEST       180         1       0
DEBILT          71         0       0
DUSSELDORF      26         0       0
HEATHROW        86         0       0
KASSEL           6         0       0
LJUBLJANA       48         0       1
MAASTRICHT       4         0       0
MADRID         331         0       0
MUNCHENB         6         0       0
OSLO             7         0       0
STOCKHOLM        2         0       0
VALENTIA         3         0       0
```

Summary:

- **Performance:** The model did not perform well, with both training and validation accuracies remaining low and loss values increasing over time.
- **Confusion Matrix:** The model was highly biased towards predicting the **BASEL** station, similar to the issues observed in previous models where predictions were skewed towards a single station.
- **Recommendation:** Changing the final activation function from sigmoid to softmax might improve the model’s ability to handle multi-class classification. Additionally, experimenting with more robust hyperparameters, such as learning rate, or modifying the architecture to include more or fewer layers, could improve the model's performance.

Final Conclusion:

After testing multiple Recurrent Neural Network (RNN) models using different architectures and hyperparameters, we observed consistent challenges with model performance. Across all models, the accuracy remained low, and the models struggled to generalize well to unseen data. Here are the key takeaways:

1. **Slow Learning and High Loss:**
   - Across all models, the loss values remained high, suggesting that the models struggled to learn effectively from the data. Despite using various combinations of activation functions (tanh, sigmoid) and architectures (CNN with Conv1D layers and LSTMs), the models did not exhibit significant improvements in loss reduction or accuracy gains.
   - For example, in Model 2 (CNN + LSTM + Sigmoid), the validation accuracy started at 6.41% and decreased to 3.70% by the 20th epoch, with loss values remaining high. In Model 3 (CNN + LSTM + Tanh), the validation accuracy dropped from 3.94% to just 0.15%, further illustrating the models' inability to generalize.
2. **Bias Towards Specific Stations:**
   - A key pattern observed in the confusion matrices is that many models became biased toward predicting a single station, most notably BASEL and OSLO. For instance, in the confusion matrix for Model 4, BASEL was correctly predicted 2909 times, but other stations like BELGRADE, BUDAPEST, and MAASTRICHT were almost always misclassified. This bias towards specific stations is a sign of overfitting or imbalance in the model's ability to differentiate between classes.
3. **Activation Functions and Model Structure:**
   - The use of activation functions like **sigmoid** in the output layer for multi-class classification may have contributed to the poor performance. Sigmoid is generally used for binary classification, whereas **softmax** might have been a better choice for this type of task.
   - The CNN and LSTM models, while suited for sequence-based data, may require further tuning in terms of the number of layers, learning rate, or regularization techniques to handle the complexity of the weather data more effectively.

Luis Gil

4. **Future Directions:**
    - **Softmax Output:** Implementing softmax activation in the final layer could improve the model's ability to handle multi-class classification more effectively.
    - **Data Normalization:** Applying further preprocessing techniques such as feature scaling or normalization might improve model performance by helping the models distinguish between the weather stations more clearly.
    - **Model Complexity:** Exploring deeper architectures, such as adding more convolutional layers or increasing the number of LSTM units, may also help the models better capture the relationships in the data.

In summary, while the models provided some level of station prediction, their overall performance remained suboptimal, particularly in distinguishing between different weather stations. Further optimization, such as using softmax activation and tuning hyperparameters, is recommended to improve the models' performance.