2.2 Complex Machine Learning Models & Keras Part 1

I used a CNN model due to its computational efficiency with large datasets and its ability to extract local temporal patterns via convolutional layers. While CNNs are typically employed for spatial data (e.g. image recognition), this experiment tested their suitability for time-series classification — specifically, for identifying weather stations from sequences of atmospheric readings. A key reason for choosing CNNs was that they run faster and require less memory compared to RNNs, making them more appropriate for my system, which otherwise struggles with heavier models like LSTMs (e.g. freezing or crashing during training). The final layer used a softmax activation function, appropriate for this multi-class classification task, as it outputs a probability distribution over the 15 mutually exclusive station classes. Three trials were conducted, varying hyperparameters including the number of epochs (5, 30, 50), batch sizes (16, 8), and hidden layer sizes (8, 32, 64):

- Trial 1 (epochs=5, hidden size=8) achieved an accuracy of 9.15%, missing 2 stations (*Stockholm* and *Sonnblick*). Despite low performance, this trial showed relatively more stable behaviour, with less divergence and more consistent loss trends, making it the most suitable baseline.
- Trial 2 (epochs=30, hidden size=32) performed worse, with accuracy dropping to 5.04%, and 3 stations (*Deblit*, *Ljubljana*, and *Maastricht*) completely unrecognised. Training was unstable, with both losses increasing and accuracy fluctuating erratically.
- Trial 3 (epochs=50, hidden size=64) showed severe divergence but surprisingly achieved the highest accuracy of 19.34%, despite failing to recognise 3 stations (*Budapest*, *Sonnblick*, and *Maastricht*). This apparent improvement likely reflects overfitting or collapse into a few dominant classes, rather than genuine learning.

These results suggest that increasing model capacity and training duration in this setup led to instability rather than improved learning, likely due to overfitting, poor weight updates, or an unsuitable learning rate. Despite none of the trials achieving satisfactory performance, Trial 1 remains the most stable configuration, offering a reasonable foundation for future tuning or alternative architectures.

Despite extensive experimentation, none of the CNN-based configurations delivered satisfactory results. A likely explanation is that the sequential nature of atmospheric data — where the temporal order of inputs is meaningful — makes it more suitable for models designed to handle sequences. In particular, Recurrent Neural Networks (RNNs) or their variants like LSTMs may be better suited for this task, as they are specifically designed to capture long-range dependencies and learn patterns over time. These architectures could provide better generalisation and learning dynamics for time-dependent classification problems like weather station recognition.

Trial 1

Input:

```python
# Define key parameters ---> start with small values to test model behaviour
epochs = 5              # Start small; increase after testing
batch_size = 16         # Reasonable size for small datasets
n_hidden = 8            # Small hidden layer to begin with (instead of 32)
```

```python
# Define dimensions
timesteps = len(X_train[0])         # 15 weather stations
input_dim = len(X_train[0][0])      # 9 observation types per station
n_classes = len(y_train[0])         # 15 output classes (stations)
```

```python
# Build the model
model = Sequential()
model.add(Conv1D(n_hidden, kernel_size=2, activation='relu', input_shape=(timesteps, input_dim)))
model.add(Dense(8, activation='relu'))           # Smaller dense layer to start with
model.add(MaxPooling1D())
model.add(Flatten())
model.add(Dense(n_classes, activation='softmax'))  # Output layer: softmax for multi-class classification
```

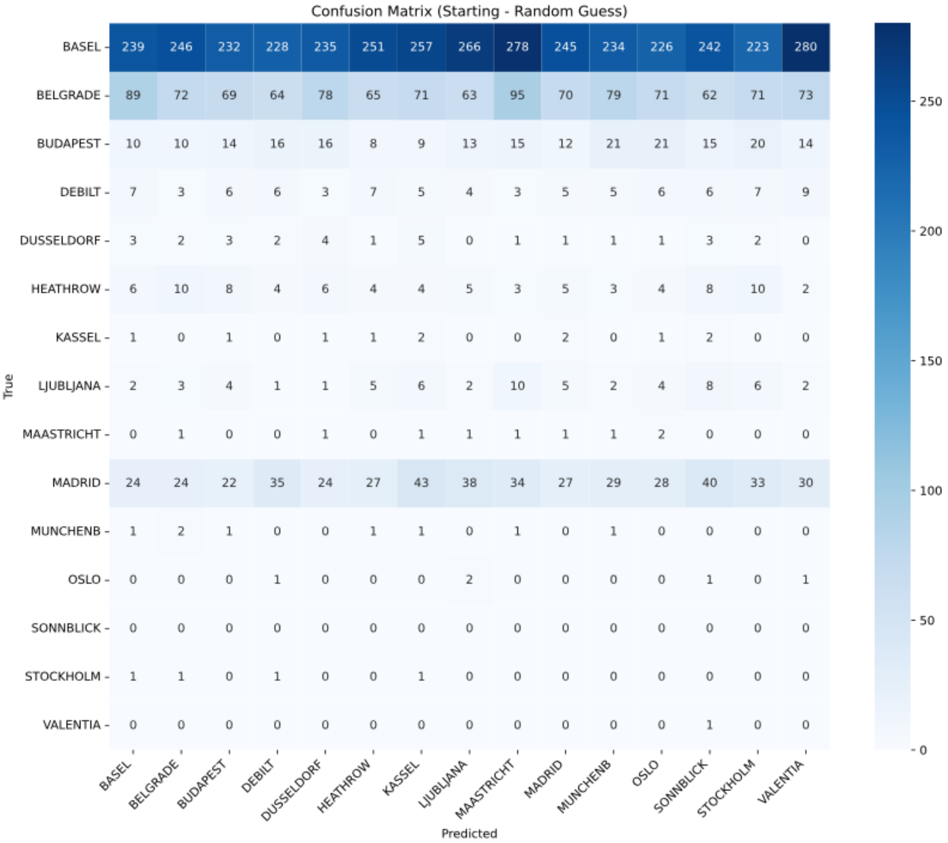
Output:

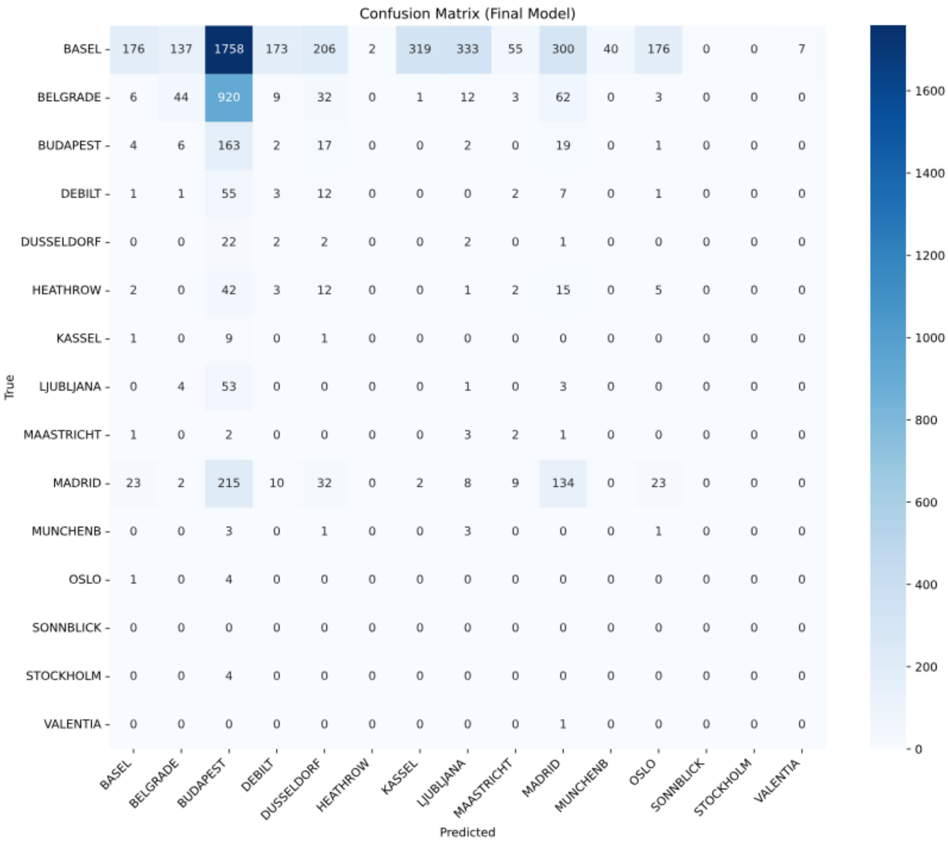
```
Epoch 1/5
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.1414 - loss: 1266.0677 - val_accuracy: 0.2321 - val_loss: 3838.2141
Epoch 2/5
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1466 - loss: 12361.7549 - val_accuracy: 0.1617 - val_loss: 21984.7695
Epoch 3/5
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1495 - loss: 40853.9258 - val_accuracy: 0.2971 - val_loss: 54569.9219
Epoch 4/5
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1441 - loss: 86181.6250 - val_accuracy: 0.1363 - val_loss: 108308.0312
Epoch 5/5
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1390 - loss: 154616.8125 - val_accuracy: 0.0915 - val_loss: 180960.6875
```

Comment:

The model is struggling to learn effectively. Training loss grows dramatically over epochs, indicating the model is diverging rather than converging. Training accuracy shows a slight increase but remains very low overall (~14-15%). Validation accuracy rises briefly early on but then falls off, demonstrating poor generalisation. These trends suggest the model is unable to capture meaningful patterns from the data and fails to improve its predictive performance over time.

Confusion Matrix





```python
from sklearn.metrics import accuracy_score

# Calculate model accuracy
# Convert true and predicted labels back to class indices
true_indices = np.argmax(y_test, axis=1)
pred_indices = np.argmax(model.predict(X_test), axis=1)

# Calculate accuracy
accuracy = accuracy_score(true_indices, pred_indices)
print(f"Model Accuracy: {accuracy * 100:.2f}%")
```

```
180/180 ━━━━━━━━━━━━━━━━━━━━ 0s 697us/step
Model Accuracy: 9.15%
```

Comment

The model was unable to recognise all 15 stations, as both *Stockholm* and *Sonnblick* were missing from its predictions. Additionally, the overall accuracy was relatively low at just 9.15%.

Trial 2

Input:

```python
# Define key parameters for improved model
epochs = 30              # Increase epochs for better learning
batch_size = 16          # Keep batch size reasonable
n_hidden = 32            # Increase hidden layer size based on previous results
```

```python
# Define dimensions (keep as before)
timesteps = len(X_train[0])          # 15 time steps (stations)
input_dim = len(X_train[0][0])       # 9 features per timestep
n_classes = len(y_train[0])          # 15 stations
```

```python
# Build model
model = Sequential()
model.add(Conv1D(n_hidden, kernel_size=2, activation='relu', input_shape=(timesteps, input_dim)))
model.add(Dense(8, activation='relu'))              # Smaller dense layer after conv
model.add(MaxPooling1D())
model.add(Flatten())
model.add(Dense(n_classes, activation='softmax'))  # Output layer: softmax for multi-class classification
```

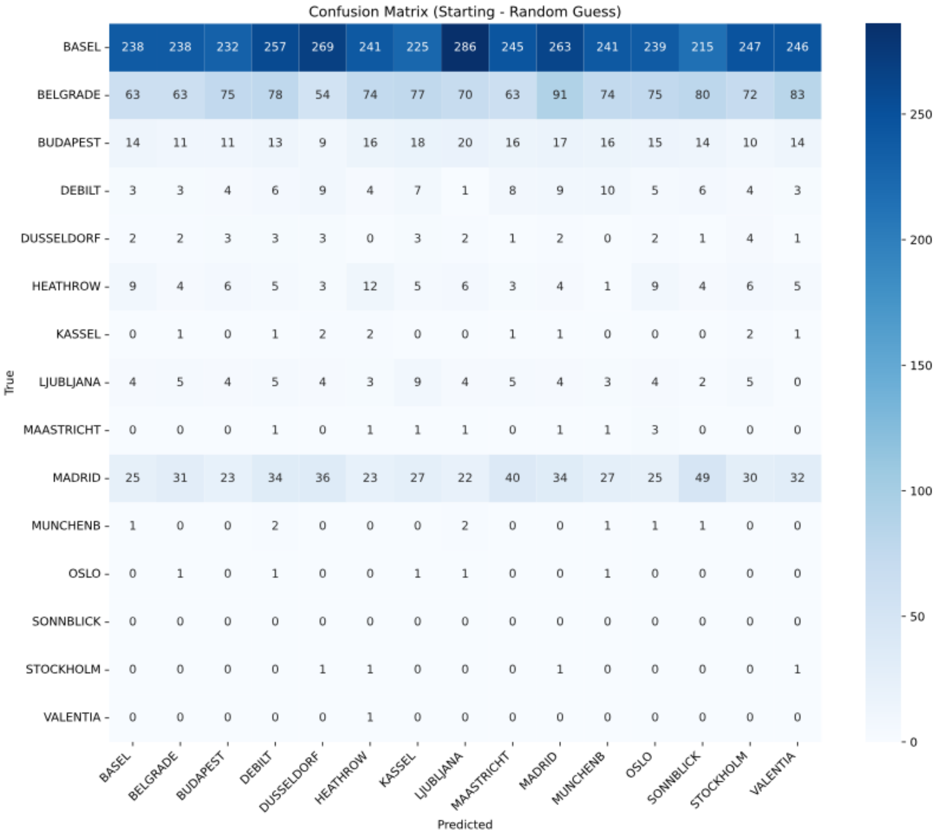
Output:

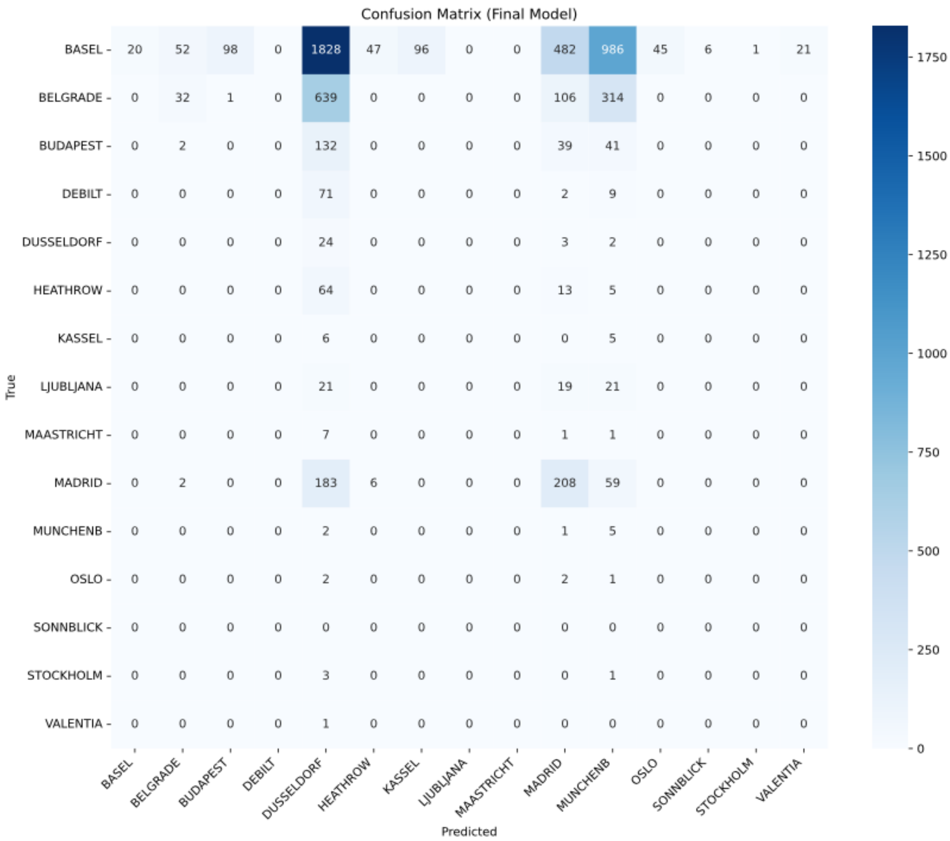
```
Epoch 1/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.1058 - loss: 4045.2676 - val_accuracy: 0.2142 - val_loss: 14689.9082
Epoch 2/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.1238 - loss: 39338.9531 - val_accuracy: 0.1131 - val_loss: 66420.1484
Epoch 3/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1273 - loss: 122024.8281 - val_accuracy: 0.1384 - val_loss: 176306.9688
Epoch 4/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.1348 - loss: 265929.0000 - val_accuracy: 0.1598 - val_loss: 338595.8125
Epoch 5/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1348 - loss: 480610.9688 - val_accuracy: 0.1046 - val_loss: 552676.1875
Epoch 6/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1343 - loss: 749638.6875 - val_accuracy: 0.2409 - val_loss: 849660.8125
Epoch 7/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.1298 - loss: 1108540.6250 - val_accuracy: 0.1039 - val_loss: 1214760.5000
Epoch 8/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.1305 - loss: 1534140.2500 - val_accuracy: 0.0896 - val_loss: 1622778.8750
Epoch 9/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1318 - loss: 2069387.3750 - val_accuracy: 0.0934 - val_loss: 2115079.7500
Epoch 10/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1267 - loss: 2621316.0000 - val_accuracy: 0.0342 - val_loss: 2724947.2500
Epoch 11/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1293 - loss: 3300527.5000 - val_accuracy: 0.0483 - val_loss: 3342549.5000
Epoch 12/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1289 - loss: 4070323.5000 - val_accuracy: 0.0486 - val_loss: 4081891.2500
Epoch 13/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1301 - loss: 4932608.0000 - val_accuracy: 0.0610 - val_loss: 4918520.0000
Epoch 14/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1272 - loss: 5866342.0000 - val_accuracy: 0.1868 - val_loss: 5832372.0000
Epoch 15/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1275 - loss: 6889095.5000 - val_accuracy: 0.1809 - val_loss: 6805226.5000
Epoch 16/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1299 - loss: 8043191.5000 - val_accuracy: 0.0722 - val_loss: 7871001.0000
Epoch 17/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1299 - loss: 9256482.0000 - val_accuracy: 0.1025 - val_loss: 9050047.0000
Epoch 18/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1283 - loss: 10579538.0000 - val_accuracy: 0.0516 - val_loss: 10321046.0000
Epoch 19/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1236 - loss: 12030875.0000 - val_accuracy: 0.0619 - val_loss: 11766686.0000
Epoch 20/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1267 - loss: 13605297.0000 - val_accuracy: 0.1333 - val_loss: 13229866.0000
Epoch 21/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1283 - loss: 15299558.0000 - val_accuracy: 0.1514 - val_loss: 14784237.0000
Epoch 22/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1250 - loss: 17087696.0000 - val_accuracy: 0.2100 - val_loss: 16526997.0000
Epoch 23/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1250 - loss: 18986990.0000 - val_accuracy: 0.0837 - val_loss: 18445468.0000
Epoch 24/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1269 - loss: 21166404.0000 - val_accuracy: 0.0565 - val_loss: 20267444.0000
Epoch 25/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1274 - loss: 23350542.0000 - val_accuracy: 0.0601 - val_loss: 22300368.0000
Epoch 26/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1214 - loss: 25668788.0000 - val_accuracy: 0.1358 - val_loss: 24567320.0000
Epoch 27/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.1270 - loss: 28131624.0000 - val_accuracy: 0.0889 - val_loss: 26941016.0000
Epoch 28/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.1231 - loss: 30878572.0000 - val_accuracy: 0.0870 - val_loss: 29426672.0000
Epoch 29/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1238 - loss: 33669876.0000 - val_accuracy: 0.2046 - val_loss: 32150724.0000
Epoch 30/30
1076/1076 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.1272 - loss: 36649704.0000 - val_accuracy: 0.0500 - val_loss: 34990548.0000
```

Comment:

The model's performance is consistently poor. Both training and validation losses increase sharply across all epochs, indicating divergence and failure to learn. Training accuracy remains very low and relatively flat (around 12-13%), showing little to no improvement. Validation accuracy is erratic, fluctuating without a clear upward trend and occasionally spiking, which suggests unstable generalisation. Overall, the model neither converges nor reliably recognises all classes, pointing to issues with model design, training process, or data representation.

Confusion Matrix





```python
from sklearn.metrics import accuracy_score

# Calculate model accuracy
# Convert true and predicted labels back to class indices
true_indices = np.argmax(y_test, axis=1)
pred_indices = np.argmax(model.predict(X_test), axis=1)

# Calculate accuracy
accuracy = accuracy_score(true_indices, pred_indices)
print(f"Model Accuracy: {accuracy * 100:.2f}%")
```

```
180/180 ━━━━━━━━━━━━━━━━━━━━ 0s 752us/step
Model Accuracy: 5.04%
```

Comment

The model was unable to recognise all 15 stations, as *Deblit*, *Ljubljana*, and *Maastricht* were missing from its predictions. Additionally, the overall accuracy was even lower at 5.04%.

Trial 3

Input

```python
# Define key parameters
epochs = 50             # Increase further to allow more learning
batch_size = 8          # Smaller batch size may help model generalise better
n_hidden = 64           # Increase model capacity to learn more complex patterns
```

```python
# Build model
model = Sequential()
model.add(Conv1D(n_hidden, kernel_size=2, activation='relu', input_shape=(timesteps, input_dim)))
model.add(Dense(8, activation='relu'))            # Smaller dense layer after conv
model.add(MaxPooling1D())
model.add(Flatten())
model.add(Dense(n_classes, activation='softmax'))  # Output layer: softmax for multi-class classification
```

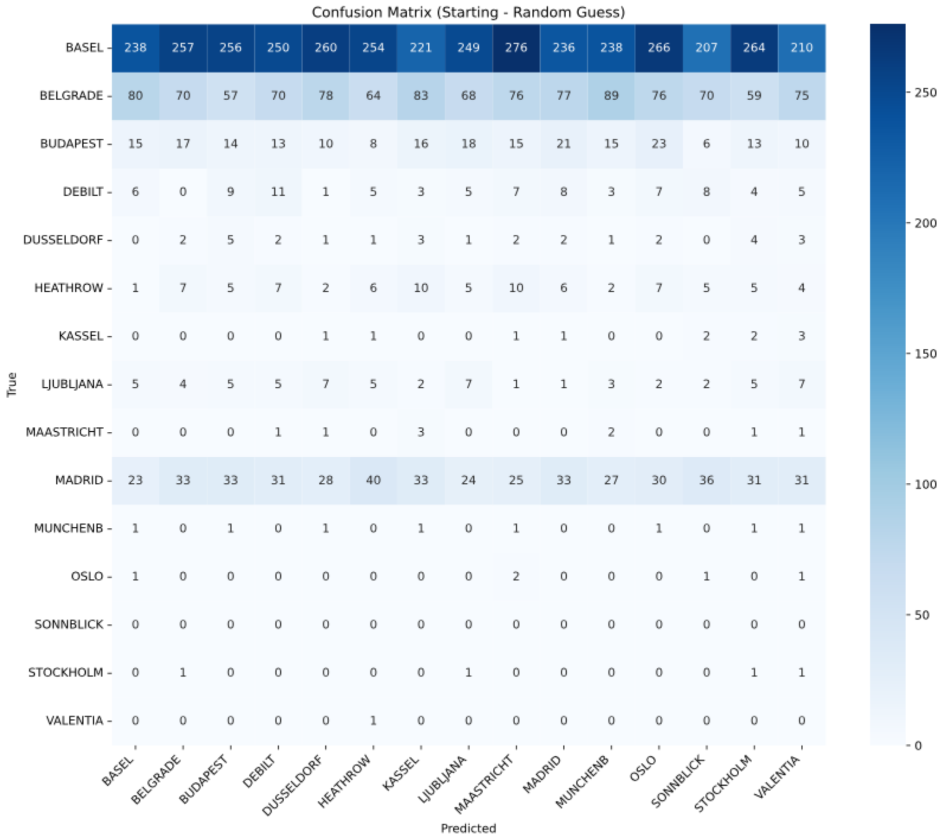
Output

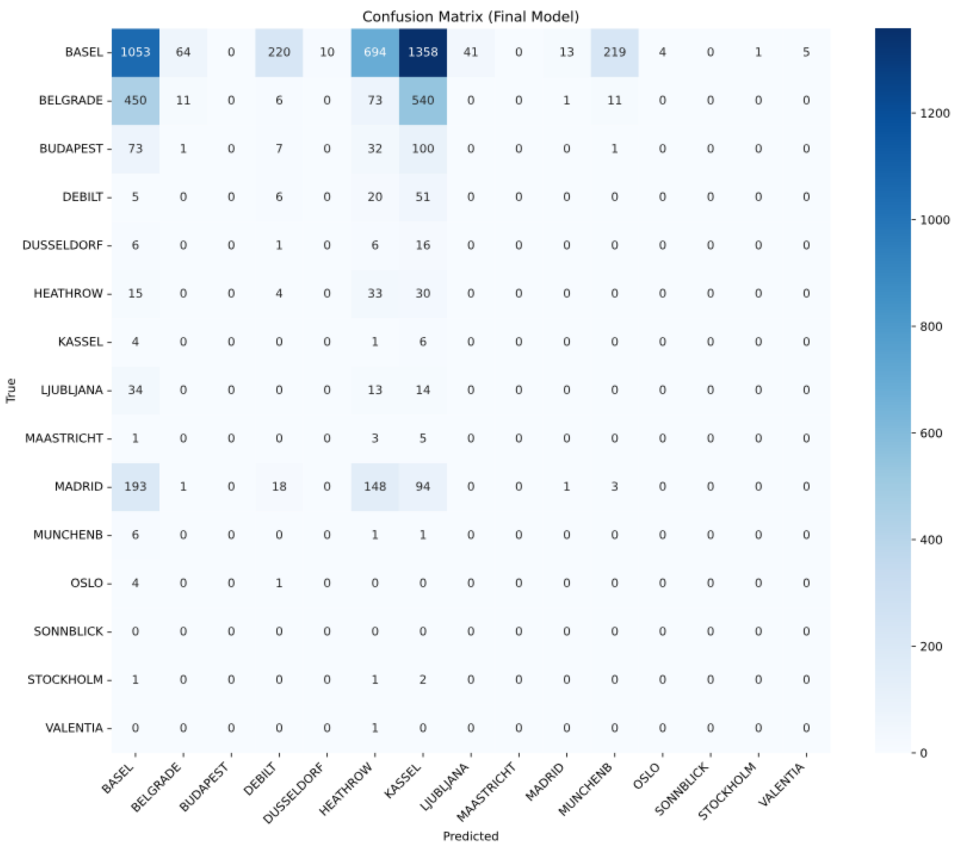
```
Epoch 1/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.1204 - loss: 57953.4180 - val_accuracy: 0.0955 - val_loss: 177353.5312
Epoch 2/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.1218 - loss: 579714.8750 - val_accuracy: 0.2062 - val_loss: 986685.0625
Epoch 3/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.1177 - loss: 1830556.3750 - val_accuracy: 0.1957 - val_loss: 2525237.2500
Epoch 4/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.1240 - loss: 3882455.2500 - val_accuracy: 0.0406 - val_loss: 4776302.5000
Epoch 5/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.1221 - loss: 6867038.5000 - val_accuracy: 0.1009 - val_loss: 7995113.5000
Epoch 6/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.1219 - loss: 10929442.0000 - val_accuracy: 0.0289 - val_loss: 12367472.0000
Epoch 7/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.1245 - loss: 16404688.0000 - val_accuracy: 0.1399 - val_loss: 17929178.0000
Epoch 8/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.1211 - loss: 23286850.0000 - val_accuracy: 0.0843 - val_loss: 24822516.0000
Epoch 9/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.1213 - loss: 31293902.0000 - val_accuracy: 0.1738 - val_loss: 33160008.0000
Epoch 10/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.1226 - loss: 41643760.0000 - val_accuracy: 0.1162 - val_loss: 43307240.0000
Epoch 11/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.1240 - loss: 53448124.0000 - val_accuracy: 0.0366 - val_loss: 55064424.0000
Epoch 12/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.1217 - loss: 67005360.0000 - val_accuracy: 0.0701 - val_loss: 68518744.0000
Epoch 13/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.1272 - loss: 83272144.0000 - val_accuracy: 0.0596 - val_loss: 84308080.0000
Epoch 14/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.1229 - loss: 101533008.0000 - val_accuracy: 0.0178 - val_loss: 102451896.0000
Epoch 15/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.1238 - loss: 122251040.0000 - val_accuracy: 0.0211 - val_loss: 122827896.0000
Epoch 16/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.1233 - loss: 146217024.0000 - val_accuracy: 0.0976 - val_loss: 145742144.0000
Epoch 17/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.1219 - loss: 172514704.0000 - val_accuracy: 0.0511 - val_loss: 170102896.0000
Epoch 18/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.1238 - loss: 201111312.0000 - val_accuracy: 0.1325 - val_loss: 198021744.0000
Epoch 19/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.1231 - loss: 234306608.0000 - val_accuracy: 0.0300 - val_loss: 229908736.0000
Epoch 20/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.1213 - loss: 268599776.0000 - val_accuracy: 0.0230 - val_loss: 264827968.0000
Epoch 21/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.1202 - loss: 309054848.0000 - val_accuracy: 0.0612 - val_loss: 300316768.0000
Epoch 22/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.1242 - loss: 349452992.0000 - val_accuracy: 0.2844 - val_loss: 341582784.0000
Epoch 23/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.1236 - loss: 395182080.0000 - val_accuracy: 0.0437 - val_loss: 384763936.0000
Epoch 24/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.1294 - loss: 444598272.0000 - val_accuracy: 0.0427 - val_loss: 432082336.0000
Epoch 25/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.1207 - loss: 499962144.0000 - val_accuracy: 0.0108 - val_loss: 482573888.0000
Epoch 26/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.1222 - loss: 556852352.0000 - val_accuracy: 0.1284 - val_loss: 536408928.0000
Epoch 27/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.1283 - loss: 619043392.0000 - val_accuracy: 0.0155 - val_loss: 595523840.0000
Epoch 28/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.1244 - loss: 685511616.0000 - val_accuracy: 0.0673 - val_loss: 659875072.0000
Epoch 29/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.1249 - loss: 754959104.0000 - val_accuracy: 0.0514 - val_loss: 724592448.0000
Epoch 30/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.1254 - loss: 833662272.0000 - val_accuracy: 0.0776 - val_loss: 796311616.0000
Epoch 31/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.1286 - loss: 909434944.0000 - val_accuracy: 0.0572 - val_loss: 872235200.0000
Epoch 32/50
2152/2152 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.1269 - loss: 997557056.0000 - val_accuracy: 0.1338 - val_loss: 950890944.0000
```

Comment

The model's performance clearly indicates severe divergence and failure to learn. Training and validation losses increase dramatically and continuously across epochs, reaching extremely high values, showing that the model is not minimising the loss but diverging instead. Although there are occasional spikes in validation accuracy, both training and validation accuracies remain consistently low and unstable throughout training, without meaningful improvement. This pattern strongly suggests fundamental issues with the model architecture, learning rate, or data preprocessing that prevent effective learning and generalisation.

Confusion Matrix






```python
from sklearn.metrics import accuracy_score

# Calculate model accuracy
# Convert true and predicted labels back to class indices
true_indices = np.argmax(y_test, axis=1)
pred_indices = np.argmax(model.predict(X_test), axis=1)

# Calculate accuracy
accuracy = accuracy_score(true_indices, pred_indices)
print(f"Model Accuracy: {accuracy * 100:.2f}%")
```

```
180/180 ━━━━━━━━━━━━━━━━━━━━ 0s 756us/step
Model Accuracy: 19.34%
```

Comment

The model was unable to recognise all 15 stations, as *Budapest*, *Sonnblick*, and *Maastricht* were missing from its predictions. However, the overall accuracy was at its greatest at 19.34%.