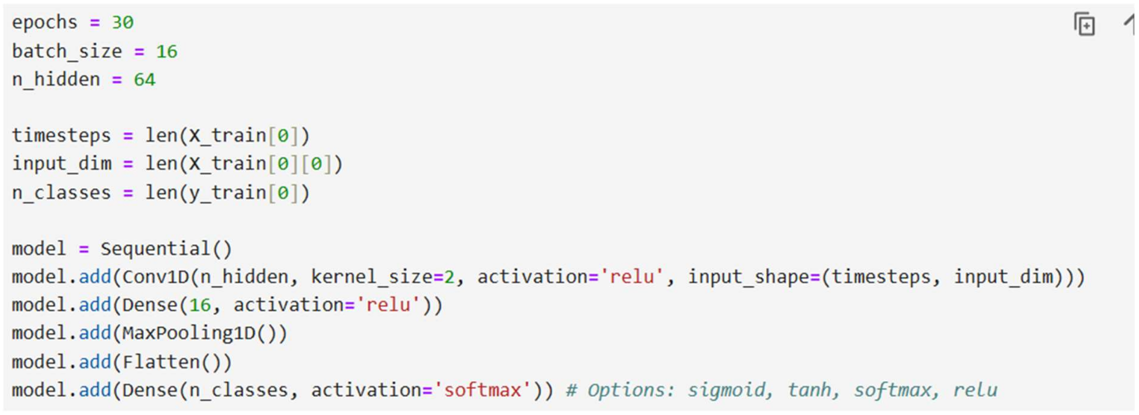
Exercise 2.2: Complex Machine Learning Models and Keras Part 1

Timothy Aluko

Section A. Convolution Neural Network (CNN) model

This section begins with running of the CNN model. The model assumed to be a better for spatial and visual realms which run quickly and handle large dataset at the same time. The model was run using different parameters and activation type as presented below.

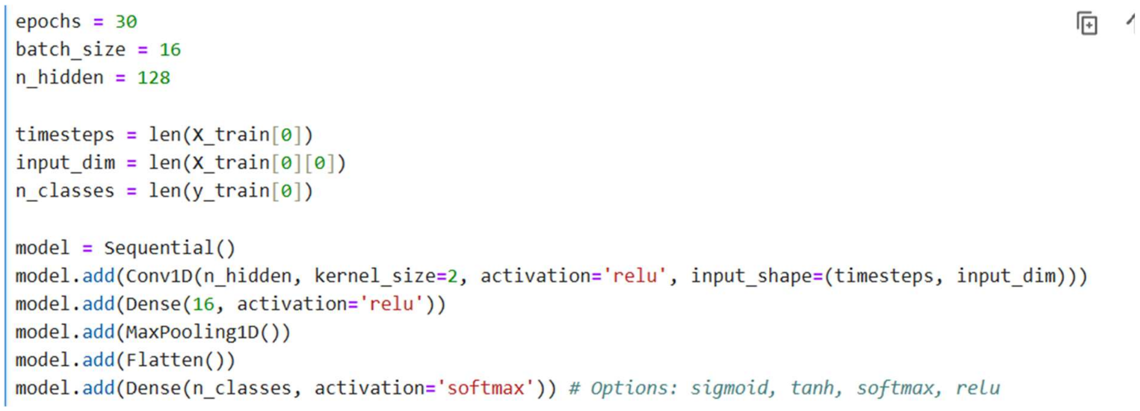


```
epochs = 30
batch_size = 16
n_hidden = 64

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='softmax')) # Options: sigmoid, tanh, softmax, relu
```

A brief summary:

- The first layer captures sequence data patterns with 64 filters in 1D convolution, producing (14, 64). Includes 1,216 parameters.
- A dense layer with 16 neurones applied over 14 timesteps refines feature extraction with 1,040 parameters. The sequence length (from 14 to 7) to focus on critical features and reduce computation.
- Easy to manage model with 3,951 trainable parameters for simple data tasks.
- Utilises convolutional and dense layers to capture sequential and dense data patterns.
- A shallow architecture allows this design to generalise on simpler datasets without overfitting.
- Add Conv1D or dense layers and dropout for increased complexity or overfitting.

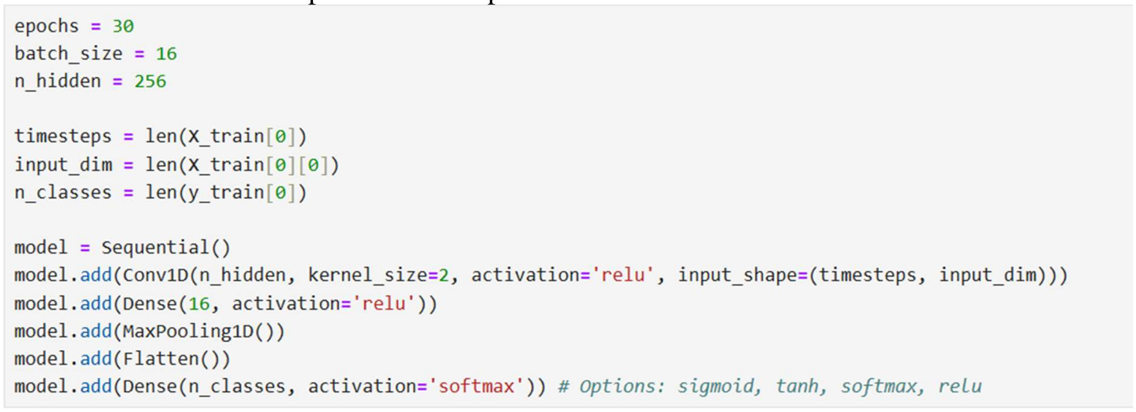
This structure works well for simpler or smaller datasets that require efficient, moderate pattern recognition without model depth. "Softmax" activation appears to be the most effective method for increasing the model's station recognition rate. What happens if we raise the hidden layers to 128?



```
epochs = 30
batch_size = 16
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='softmax')) # Options: sigmoid, tanh, softmax, relu
```

The neural network "sequential_2," with convolutional and dense layers, is structured as follows:

- The 128-filter Conv1D Layer improves pattern recognition over 14 timesteps with 2,432 parameters.
- Added 2,064 parameters to dense layer with 16 neurones, refining features at each timestep.
- MaxPooling1D cuts sequence length in half (from 14 to 7), simplifying data processing.
- The Flatten Layer transforms data into a 112-dimensional vector for classification.
- The final output dense layer, with 15 stations, is likely for multi-class classification.
- With 6,191 trainable parameters, it is moderately complex and efficient for handling data patterns.
- Balanced Complexity (6,191 parameters) balances performance and generalisation for moderate datasets.
- The time required to get the output for each epoch is proportional to M.
- The model is able to identify all 15 weather stations when the number of layers is raised to 256.
- There is a lack of precision:12.4 percent

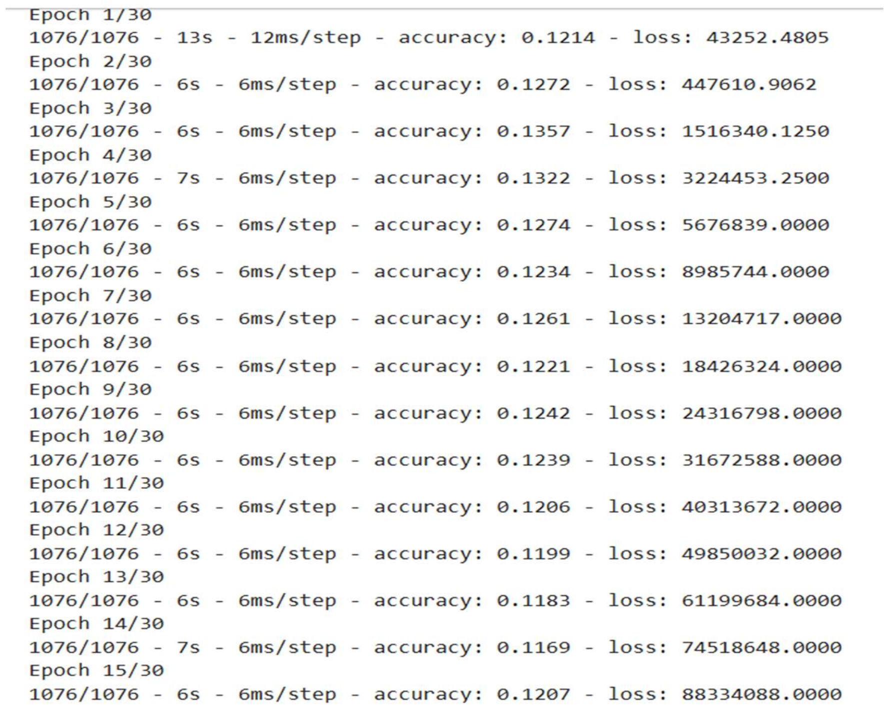


```
epochs = 30
batch_size = 16
n_hidden = 256

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='softmax')) # Options: sigmoid, tanh, softmax, relu
```

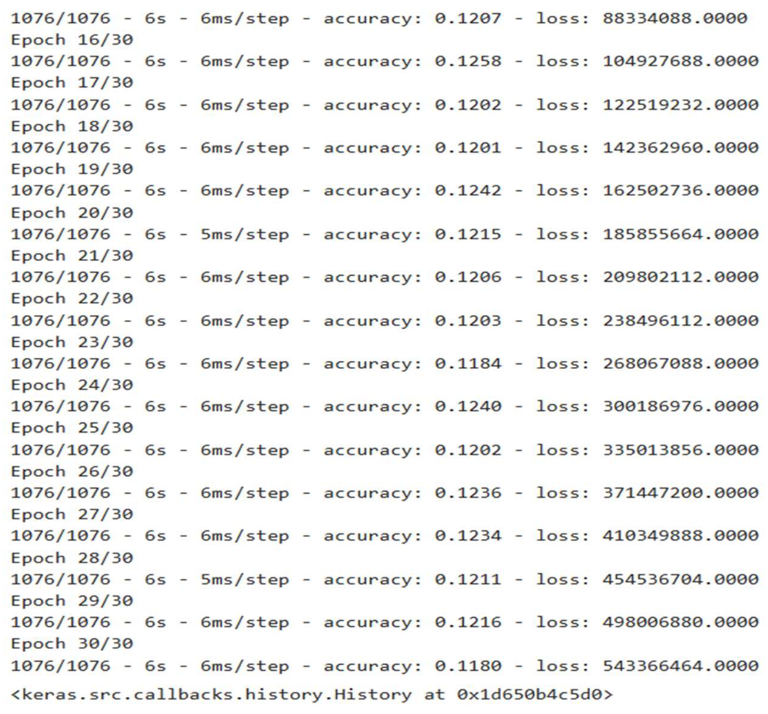
Its moderately complex structure with convolutional and dense layers makes model "sequential_3" suitable for feature extraction and classification:

- Conv1D Layer: Detects detailed patterns with 256 filters and 4,864 parameters.
- The dense layer adds 16 neurones to refine feature representation, adding 4,112 parameters. MaxPooling1D: Shortens sequences for efficiency.
- Flattening prepares data for the final 15-unit output layer, likely for multi-class classification.
- The model handles moderate data complexity with 10,671 trainable parameters.

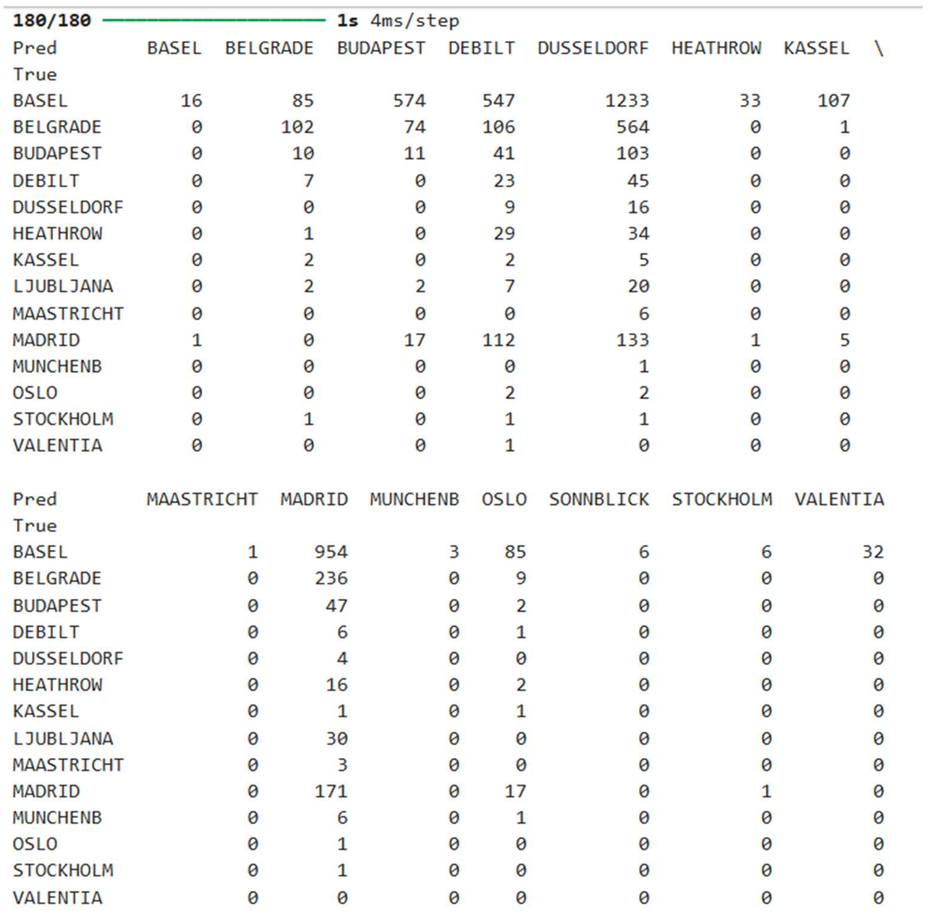
This structured layering model efficiently processes and classifies moderate datasets for feature extraction and classification applications.



```
Epoch 1/30
1076/1076 - 13s - 12ms/step - accuracy: 0.1214 - loss: 43252.4805
Epoch 2/30
1076/1076 - 6s - 6ms/step - accuracy: 0.1272 - loss: 447610.9062
Epoch 3/30
1076/1076 - 6s - 6ms/step - accuracy: 0.1357 - loss: 1516340.1250
Epoch 4/30
1076/1076 - 7s - 6ms/step - accuracy: 0.1322 - loss: 3224453.2500
Epoch 5/30
1076/1076 - 6s - 6ms/step - accuracy: 0.1274 - loss: 5676839.0000
Epoch 6/30
1076/1076 - 6s - 6ms/step - accuracy: 0.1234 - loss: 8985744.0000
Epoch 7/30
1076/1076 - 6s - 6ms/step - accuracy: 0.1261 - loss: 13204717.0000
Epoch 8/30
1076/1076 - 6s - 6ms/step - accuracy: 0.1221 - loss: 18426324.0000
Epoch 9/30
1076/1076 - 6s - 6ms/step - accuracy: 0.1242 - loss: 24316798.0000
Epoch 10/30
1076/1076 - 6s - 6ms/step - accuracy: 0.1239 - loss: 31672588.0000
Epoch 11/30
1076/1076 - 6s - 6ms/step - accuracy: 0.1206 - loss: 40313672.0000
Epoch 12/30
1076/1076 - 6s - 6ms/step - accuracy: 0.1199 - loss: 49850032.0000
Epoch 13/30
1076/1076 - 6s - 6ms/step - accuracy: 0.1183 - loss: 61199684.0000
Epoch 14/30
1076/1076 - 7s - 6ms/step - accuracy: 0.1169 - loss: 74518648.0000
Epoch 15/30
1076/1076 - 6s - 6ms/step - accuracy: 0.1207 - loss: 88334088.0000
```



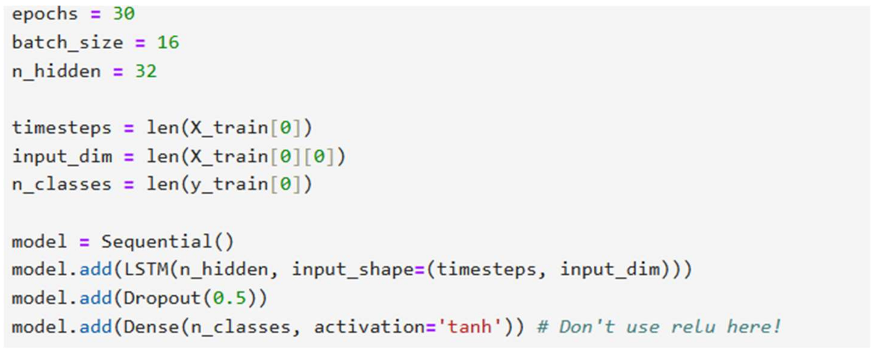
```
1076/1076 - 6s - 6ms/step - accuracy: 0.1207 - loss: 88334088.0000
Epoch 16/30
1076/1076 - 6s - 6ms/step - accuracy: 0.1258 - loss: 104927688.0000
Epoch 17/30
1076/1076 - 6s - 6ms/step - accuracy: 0.1202 - loss: 122519232.0000
Epoch 18/30
1076/1076 - 6s - 6ms/step - accuracy: 0.1201 - loss: 142362960.0000
Epoch 19/30
1076/1076 - 6s - 6ms/step - accuracy: 0.1242 - loss: 162502736.0000
Epoch 20/30
1076/1076 - 6s - 5ms/step - accuracy: 0.1215 - loss: 185855664.0000
Epoch 21/30
1076/1076 - 6s - 6ms/step - accuracy: 0.1206 - loss: 209802112.0000
Epoch 22/30
1076/1076 - 6s - 6ms/step - accuracy: 0.1203 - loss: 238496112.0000
Epoch 23/30
1076/1076 - 6s - 6ms/step - accuracy: 0.1184 - loss: 268067088.0000
Epoch 24/30
1076/1076 - 6s - 6ms/step - accuracy: 0.1240 - loss: 300186976.0000
Epoch 25/30
1076/1076 - 6s - 6ms/step - accuracy: 0.1202 - loss: 335013856.0000
Epoch 26/30
1076/1076 - 6s - 6ms/step - accuracy: 0.1236 - loss: 371447200.0000
Epoch 27/30
1076/1076 - 6s - 6ms/step - accuracy: 0.1234 - loss: 410349888.0000
Epoch 28/30
1076/1076 - 6s - 5ms/step - accuracy: 0.1211 - loss: 454536704.0000
Epoch 29/30
1076/1076 - 6s - 6ms/step - accuracy: 0.1216 - loss: 498006880.0000
Epoch 30/30
1076/1076 - 6s - 6ms/step - accuracy: 0.1180 - loss: 543366464.0000
<keras.src.callbacks.history.History at 0x1d650b4c5d0>
```



```
180/180 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
Pred        BASEL  BELGRADE  BUDAPEST  DEBILT  DUSSELDORF  HEATHROW  KASSEL  \
True
BASEL          16        85       574     547        1233        33     107
BELGRADE        0       102        74     106         564         0       1
BUDAPEST        0        10        11      41         103         0       0
DEBILT          0         7         0      23          45         0       0
DUSSELDORF      0         0         0       9          16         0       0
HEATHROW        0         1         0      29          34         0       0
KASSEL          0         2         0       2           5         0       0
LJUBLJANA       0         2         2       7          20         0       0
MAASTRICHT      0         0         0       0           6         0       0
MADRID          1         0        17     112         133         1       5
MUNCHENB        0         0         0       0           1         0       0
OSLO            0         0         0       2           2         0       0
STOCKHOLM       0         1         0       1           1         0       0
VALENTIA        0         0         0       1           0         0       0

Pred        MAASTRICHT  MADRID  MUNCHENB  OSLO  SONNBLICK  STOCKHOLM  VALENTIA
True
BASEL                1     954         3    85          6          6        32
BELGRADE             0     236         0     9          0          0         0
BUDAPEST             0      47         0     2          0          0         0
DEBILT               0       6         0     1          0          0         0
DUSSELDORF           0       4         0     0          0          0         0
HEATHROW             0      16         0     2          0          0         0
KASSEL               0       1         0     1          0          0         0
LJUBLJANA            0      30         0     0          0          0         0
MAASTRICHT           0       3         0     0          0          0         0
MADRID               0     171         0    17          0          1         0
MUNCHENB             0       6         0     1          0          0         0
OSLO                 0       1         0     0          0          0         0
STOCKHOLM            0       1         0     0          0          0         0
VALENTIA             0       0         0     0          0          0         0
```

Section B. Recurrent Neural Network (RNN) / LSTM model

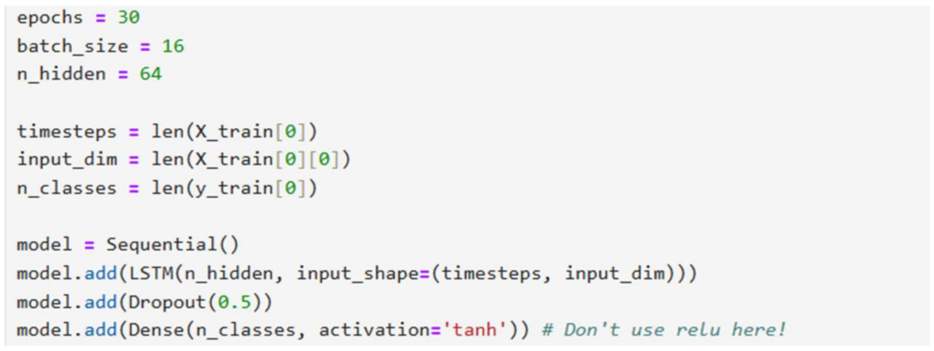
Here, the RNN model was run using different parameters and activation types.



```python
epochs = 30
batch_size = 16
n_hidden = 32

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')) # Don't use relu here!
```

- LSTM Layer: Captures sequential dependencies with 5,376 parameters.
- Dropout Layer: Randomly deactivates neurones during training to prevent overfitting.
- Dense Output Layer: 15 classification units, 495 parameters.

The model's 5,871 trainable parameters make it a streamlined, moderately complex model for sequence pattern recognition. The model seems to be able to recognise the most stations when "tanh" is turned on. But, let's see what happens when we add 64 more hidden layers.



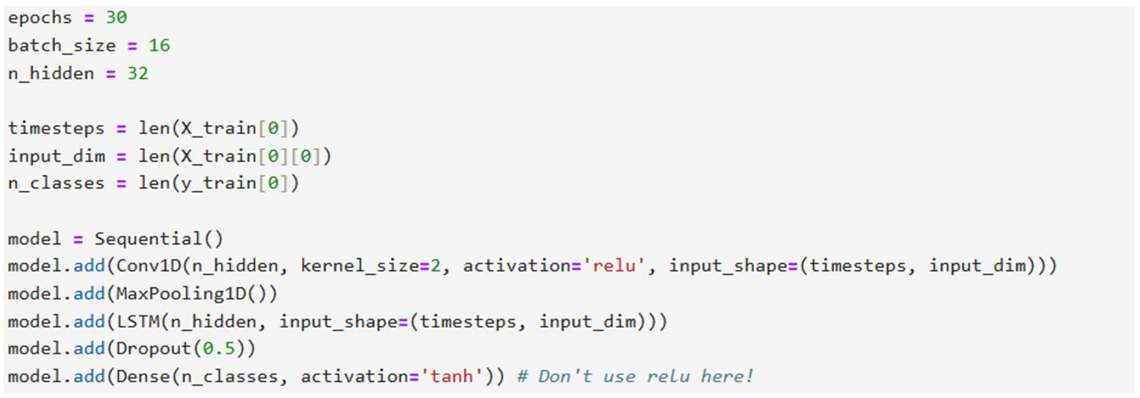
```
epochs = 30
batch_size = 16
n_hidden = 64

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')) # Don't use relu here!
```

Sequential LSTM architecture used in this model:

- LSTM Layer: Highly capable of sequential pattern learning with 18,944 parameters.
- Dropout Layer: Randomly drops connections during training to reduce overfitting.
- Dense Output Layer: 15 units with 975 final classification parameters.

All trainable parameters: 19,919 This moderately complex model emphasises sequence dependencies for classification.

But let us consider the impact of adding Conv1D and MaxPooling layers to the model.

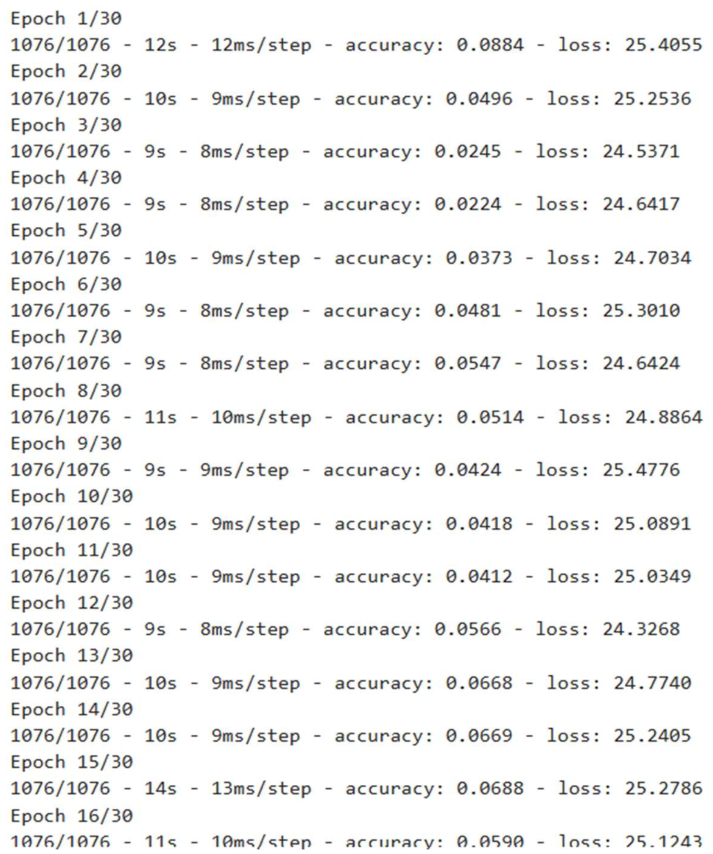


```
epochs = 30
batch_size = 16
n_hidden = 32

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(MaxPooling1D())
model.add(LSTM(n_hidden, input_shape=(timesteps, input_dim)))
model.add(Dropout(0.5))
model.add(Dense(n_classes, activation='tanh')) # Don't use relu here!
```

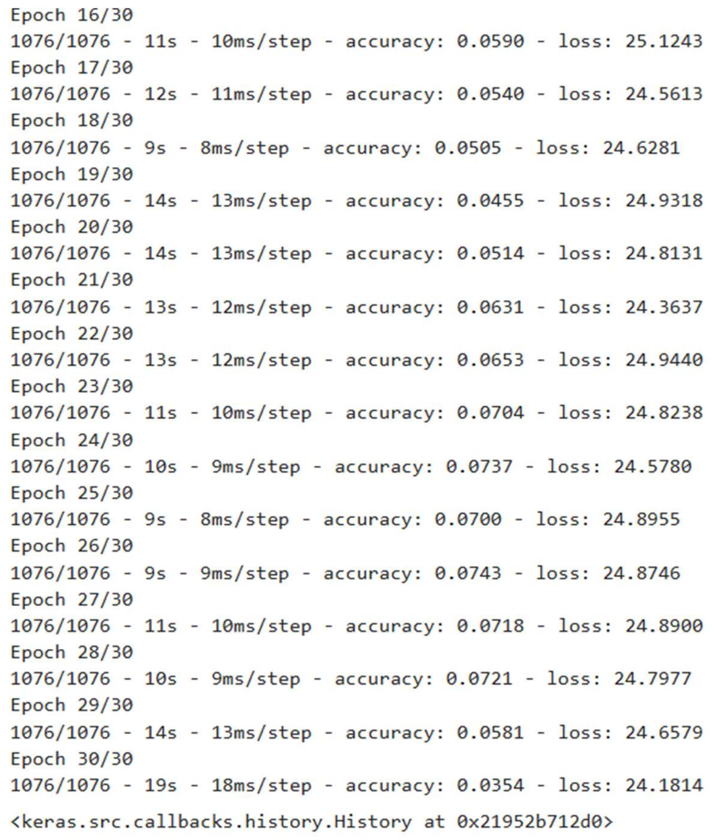
The LSTM model takes longer to run when compared to the CNN model. However, it is efficiently process sequential data, this model uses convolutional, max-pooling, LSTM, dropout, and dense layers:

- Conv1D Layer detects patterns is with 32 filters and 608 parameters.
- MaxPooling1D is half-sequence length, reduces computation.
- 8,320-parameter LSTM Layer captures sequence dependencies.
- Dropout shows the model deactivates 50% of LSTM units during training to reduce overfitting.
- The dense output layer shows 15 classification units and 495 parameters.

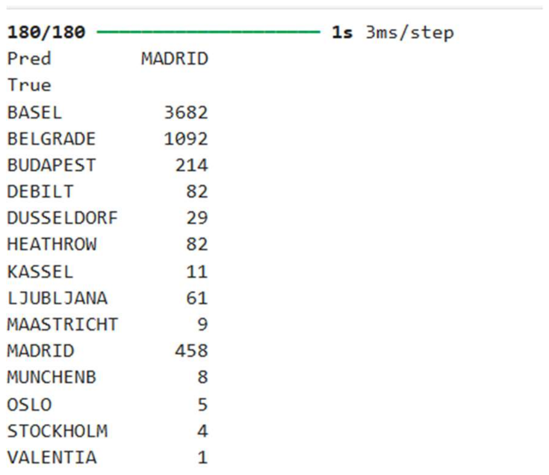
Total parameters: 0.035 (trainable), making it an efficient time series classification model.



```
Epoch 1/30
1076/1076 - 12s - 12ms/step - accuracy: 0.0884 - loss: 25.4055
Epoch 2/30
1076/1076 - 10s - 9ms/step - accuracy: 0.0496 - loss: 25.2536
Epoch 3/30
1076/1076 - 9s - 8ms/step - accuracy: 0.0245 - loss: 24.5371
Epoch 4/30
1076/1076 - 9s - 8ms/step - accuracy: 0.0224 - loss: 24.6417
Epoch 5/30
1076/1076 - 10s - 9ms/step - accuracy: 0.0373 - loss: 24.7034
Epoch 6/30
1076/1076 - 9s - 8ms/step - accuracy: 0.0481 - loss: 25.3010
Epoch 7/30
1076/1076 - 9s - 8ms/step - accuracy: 0.0547 - loss: 24.6424
Epoch 8/30
1076/1076 - 11s - 10ms/step - accuracy: 0.0514 - loss: 24.8864
Epoch 9/30
1076/1076 - 9s - 9ms/step - accuracy: 0.0424 - loss: 25.4776
Epoch 10/30
1076/1076 - 10s - 9ms/step - accuracy: 0.0418 - loss: 25.0891
Epoch 11/30
1076/1076 - 10s - 9ms/step - accuracy: 0.0412 - loss: 25.0349
Epoch 12/30
1076/1076 - 9s - 8ms/step - accuracy: 0.0566 - loss: 24.3268
Epoch 13/30
1076/1076 - 10s - 9ms/step - accuracy: 0.0668 - loss: 24.7740
Epoch 14/30
1076/1076 - 10s - 9ms/step - accuracy: 0.0669 - loss: 25.2405
Epoch 15/30
1076/1076 - 14s - 13ms/step - accuracy: 0.0688 - loss: 25.2786
Epoch 16/30
1076/1076 - 11s - 10ms/step - accuracy: 0.0590 - loss: 25.1243
```



```
Epoch 16/30
1076/1076 - 11s - 10ms/step - accuracy: 0.0590 - loss: 25.1243
Epoch 17/30
1076/1076 - 12s - 11ms/step - accuracy: 0.0540 - loss: 24.5613
Epoch 18/30
1076/1076 - 9s - 8ms/step - accuracy: 0.0505 - loss: 24.6281
Epoch 19/30
1076/1076 - 14s - 13ms/step - accuracy: 0.0455 - loss: 24.9318
Epoch 20/30
1076/1076 - 14s - 13ms/step - accuracy: 0.0514 - loss: 24.8131
Epoch 21/30
1076/1076 - 13s - 12ms/step - accuracy: 0.0631 - loss: 24.3637
Epoch 22/30
1076/1076 - 13s - 12ms/step - accuracy: 0.0653 - loss: 24.9440
Epoch 23/30
1076/1076 - 11s - 10ms/step - accuracy: 0.0704 - loss: 24.8238
Epoch 24/30
1076/1076 - 10s - 9ms/step - accuracy: 0.0737 - loss: 24.5780
Epoch 25/30
1076/1076 - 9s - 8ms/step - accuracy: 0.0700 - loss: 24.8955
Epoch 26/30
1076/1076 - 9s - 9ms/step - accuracy: 0.0743 - loss: 24.8746
Epoch 27/30
1076/1076 - 11s - 10ms/step - accuracy: 0.0718 - loss: 24.8900
Epoch 28/30
1076/1076 - 10s - 9ms/step - accuracy: 0.0721 - loss: 24.7977
Epoch 29/30
1076/1076 - 14s - 13ms/step - accuracy: 0.0581 - loss: 24.6579
Epoch 30/30
1076/1076 - 19s - 18ms/step - accuracy: 0.0354 - loss: 24.1814
<keras.src.callbacks.history.History at 0x21952b712d0>
```



```
180/180 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
Pred        MADRID
True
BASEL         3682
BELGRADE      1092
BUDAPEST       214
DEBILT          82
DUSSELDORF      29
HEATHROW        82
KASSEL          11
LJUBLJANA       61
MAASTRICHT       9
MADRID         458
MUNCHENB         8
OSLO             5
STOCKHOLM        4
VALENTIA         1
```

The model report can be concluded as follows, considering the application of the RNN model:

- CNN is faster, but the model takes longer to run.
- Things work better when the tanh is activated.
- When you add the Conv1D and MaxPooling layers, the accuracy rate goes up, but it's still not as good as CNN.
- It's a lot less than what CNN reports.
- The model can't figure out all 15 weather stations.