Susan Wang

Complex Machine Learning Models and Keras Part I

Contents:

Script I & II: Cleaning process (unscaled and scaled)

- Loaded weather data and answers data.
- Dropped date and month columns.
- Removed the three stations not in answers data.
- Several stations are missing the following observations: wind_speed and snow_depth. Columns with these observations have been removed.
- The following data are missing and filled by imputation of values from neighboring stations:
    - Kassel cloud_cover imputed with Ljubljana cloud_cover
    - Munchenb pressure imputed with Sonnblick pressure
    - Stockholm humidity imputed with Oslo humidity

- Final shape of cleaned weather data: (22950, 135)
- Final shape of answers data: (22950, 15)

Script III: Running RNN model in Keras – Unscaled Data

I have chosen the RNN model. This model might work better for this data because there is a time component involved in the data set. The observations have been recorded over 60 years, and we wish to use the model to predict weather in the future.

Preparation steps:

- New script for RNN
- Loaded the data set
- Reshape data set to (22950, 15, 9)

Scenarios for running RNN models (unscaled data):

Hyperparameters

Scenario	epochs	batch_size	n_hidden	activation
1	30	16	8	sigmoid
2	30	32	16	sigmoid
3	30	32	32	sigmoid
4	30	32	32	softmax
5	30	8	32	sigmoid
6	30	16	8	tanh
7	30	64	32	sigmoid
8	60	64	32	sigmoid
9	30	32	64	sigmoid
10	20	32	8	sigmoid
11	10	16	32	sigmoid

Results

Scenario	Final Accuracy	Final Loss	Comments
1	incr. to 0.597	incr. to 13.438	not converging, loss increasing
2	incr. to 0.641	stay 11.033	not converging, loss constant

3	incr. to 0.644	stay 11.462	not converging, loss constant
4	incr. to 0.071	incr. to 10.973	bad accuracy
5	incr. to 0.644	incr. to 22.740	not converging, high loss
6	incr. to 0.644	nan	loss nan
7	incr. to 0.644	incr. to 11.424	best of acc vs loss so far
8	incr. to 0.644	incr. to 12.832	more loss with more epochs
9	incr. to 0.644	incr. to 15.311	high loss
10	incr. to 0.322	incr. to 9.811	min loss so far, poor accuracy
11	incr. to 0.644	incr. to 10.953	improved acc vs loss

Screenshot of final Keras model layout

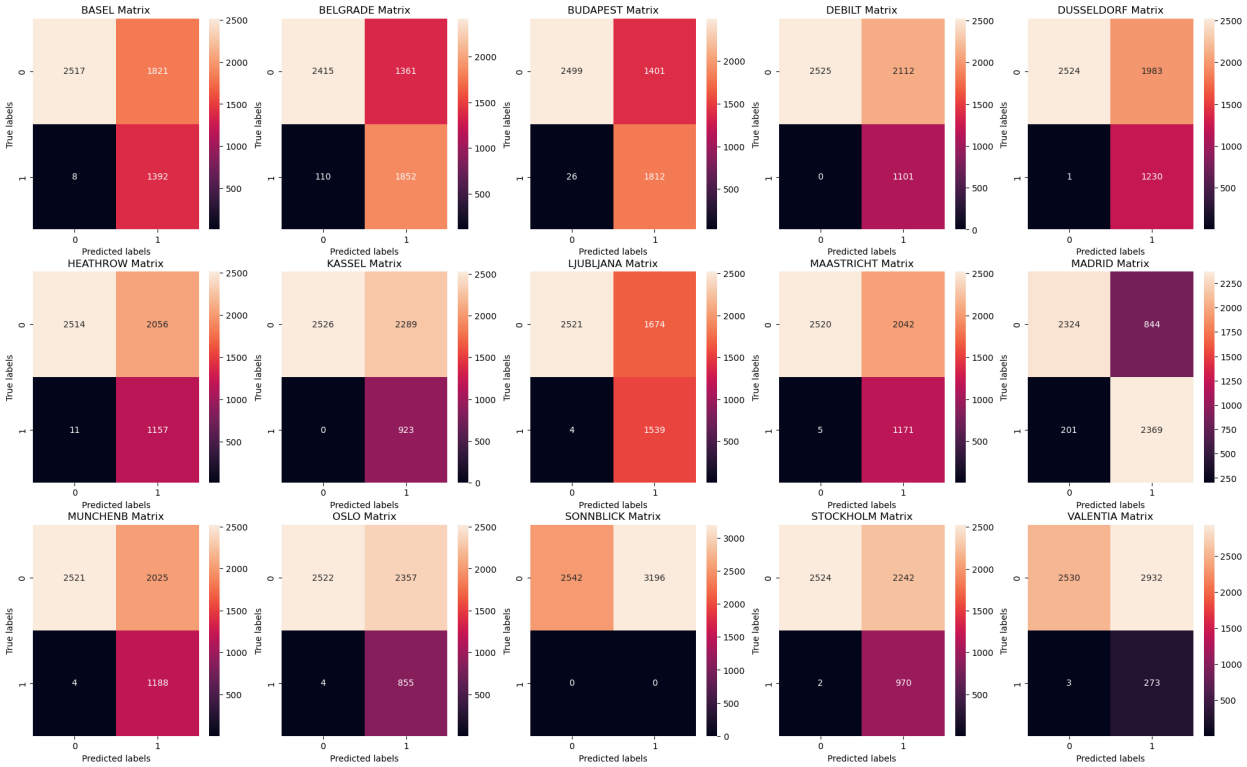
[89]:
```
epochs = 10
batch_size =16
n_hidden = 32

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

model = Sequential([
    Input(shape=(timesteps, input_dim)),
    LSTM(n_hidden),
    Dropout(0.5),
    Dense(n_classes, activation='sigmoid') ])
```

[90]:
```
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

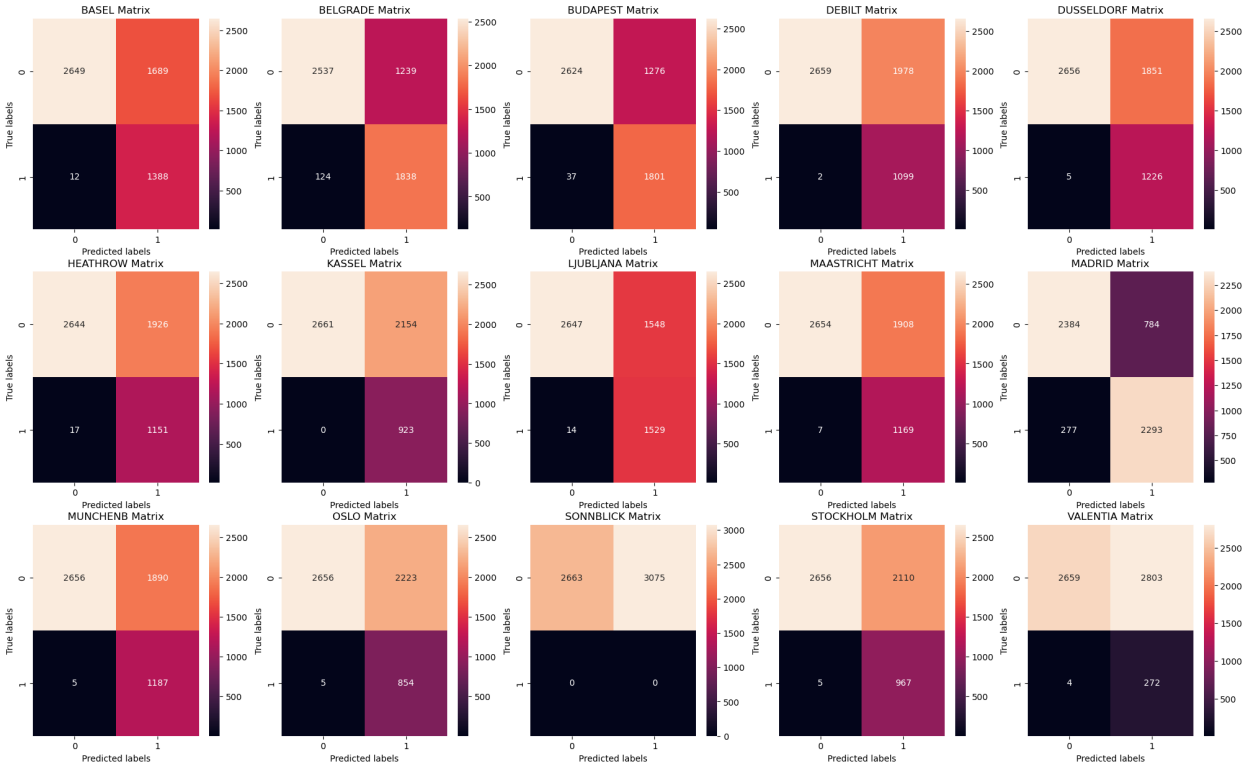
[91]:
```
model.fit(X_train,
          y_train,
          batch_size=batch_size,
          validation_data=(X_test, y_test),
          epochs=epochs)
```

Confusion Matrices RNN

Starting Confusion Matrix (Scenario 1)



Final Confusion Matrix (Scenario 11)



Script IV: Running RNN model in Keras – Scaled Data

Scenarios for running RNN models (scaled data):

Hyperparameters

Scenario	epochs	batch_size	n_hidden	activation
1	30	16	8	sigmoid
2	10	16	32	sigmoid
3	20	16	32	sigmoid
4	30	64	32	sigmoid
5	30	64	64	sigmoid
6	20	32	64	sigmoid
7	20	64	64	sigmoid
8	20	64	128	sigmoid

Results

Scenario	Final Accuracy	Final Loss	Comments
1	incr. to 0.598	incr. to 13.577	not converging, loss increasing
2	incr. to 0.647	incr. to 11.934	higher acc than unscaled
3	incr. to 0.646	incr. to 13.045	accuracy lowers and loss increases after 10 epochs
4	incr. to 0.645	incr. to 10.711	best so far
5	incr. to 0.647	incr. to 11.387	acc plateaus around 18 epochs
6	incr. to 0.646	incr. to 11.773	highest acc at epoch 12
7	incr. to 0.646	incr. to 11.077	diff results than scen. 5
8	incr. to 0.644	incr. to 14.028	high loss

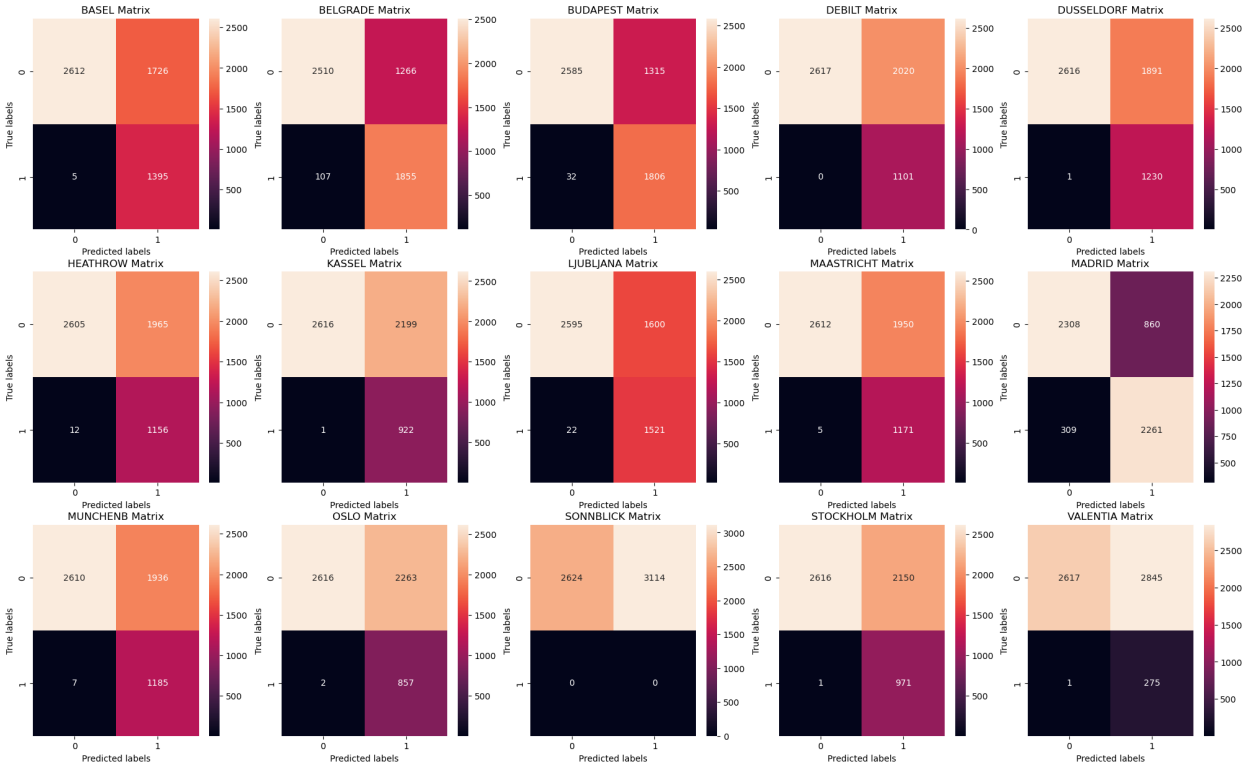
Note: On running the 'optimal' hyperparameters (scen. 4) a second time, the results were not as good as the first run: Accuracy was at 0.644, loss at 10.967.

Screenshot of final Keras model layout

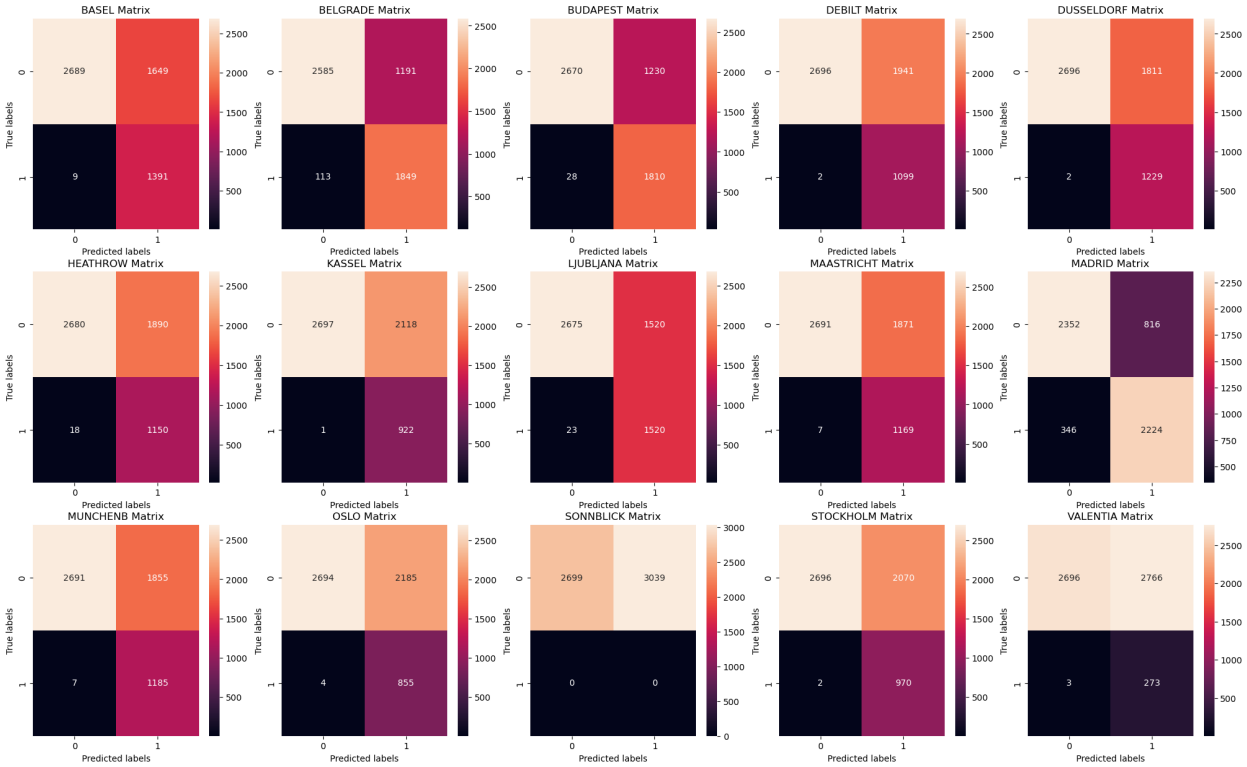
```
[52]: epochs = 30
      batch_size =64
      n_hidden = 32

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

      model = Sequential([
          Input(shape=(timesteps, input_dim)),
          LSTM(n_hidden),
          Dropout(0.5),
          Dense(n_classes, activation='sigmoid') ])

[53]: model.compile(loss='categorical_crossentropy',
                    optimizer='rmsprop',
                    metrics=['accuracy'])

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

Confusion Matrices RNN - Scaled Data

Starting Confusion Matrix (Scenario 1)



Final Confusion Matrix (Scenario 4)



Script V: Running CNN model in Keras – Scaled Data

Scenarios for running CNN models (scaled data):

Hyperparameters

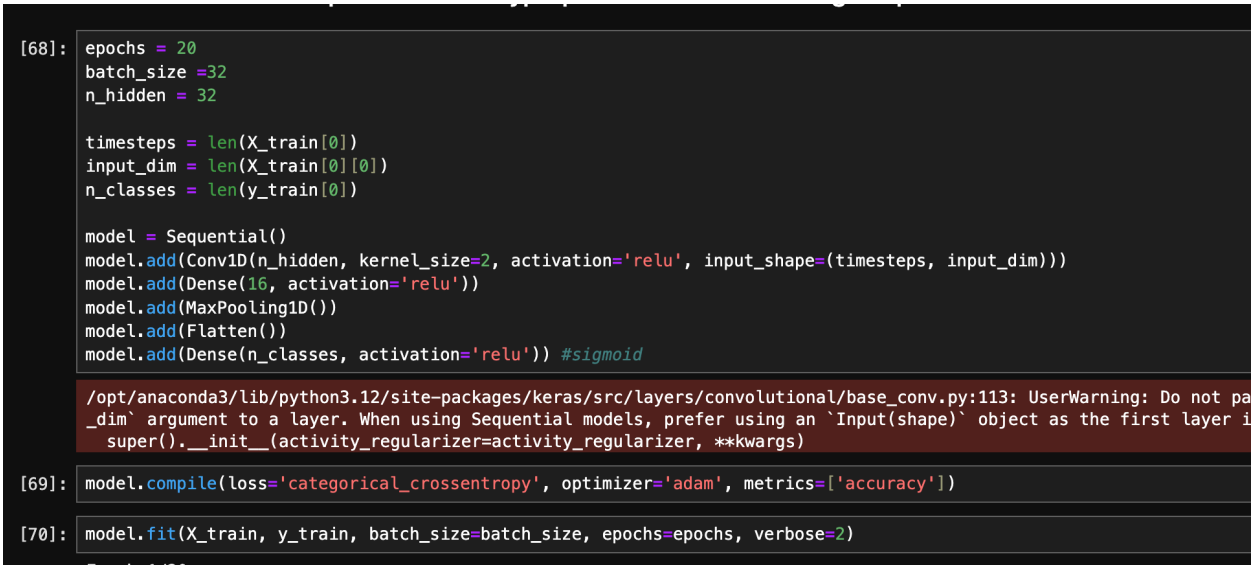
Scenario	epochs	batch_size	n_hidden	activation
1	30	16	32	softmax
2	30	16	32	sigmoid
3	20	16	8	relu
4	15	16	16	relu
5	10	8	16	relu
6	10	32	16	relu
7	20	32	16	relu
8	20	64	32	relu
9	20	32	32	relu
10	20	16	64	relu

Results

Scenario	Final Accuracy	Final Loss	Comments
1	incr. to 0.125	incr. expon.	loss very large, try another activation
2	incr. to 0.644	incr. expon.	acc plateaus after epoch 4, loss incr. exponentially
3	incr. to 0.644	10.32 - nan	better loss, but turns into nan
4	incr. to 0.644	9.63 - nan	better loss
5	incr. to 0.644	12.362 - nan	high loss
6	incr. to 0.125	decr. To 7.728	best loss yet, more epochs to improve acc
7	incr. to 0.644	9.807 - nan	more epochs = nan
8	incr. to 0.644	8.223 - nan	

9	incr. to 0.644	7.674 - nan	highest acc with lowest loss
10	incr. to 0.644	10.280 – nan	plateau at 4 epochs

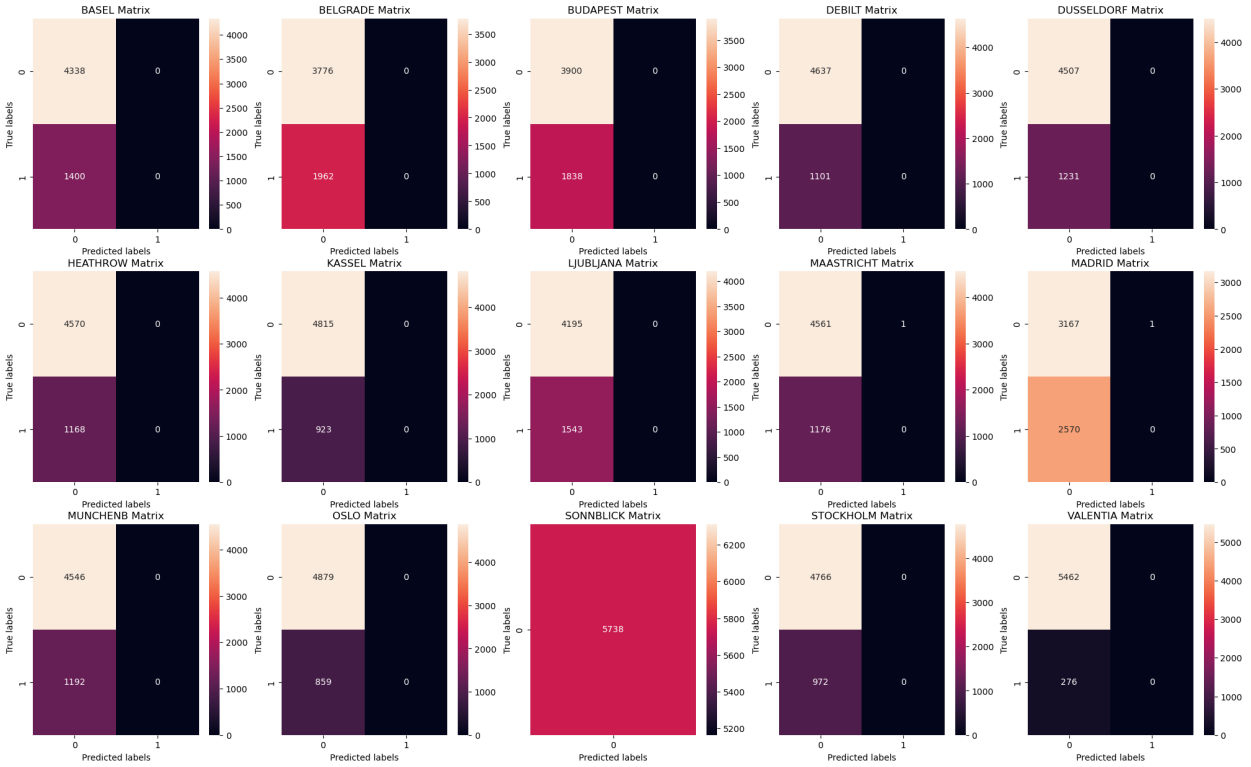
Screenshot of final Keras model layout



```
[68]: epochs = 20
      batch_size =32
      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(Dense(16, activation='relu'))
      model.add(MaxPooling1D())
      model.add(Flatten())
      model.add(Dense(n_classes, activation='relu')) #sigmoid

      /opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pa
      _dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer i
        super().__init__(activity_regularizer=activity_regularizer, **kwargs)

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

[70]: model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs, verbose=2)
```

Confusion Matrices CNN – Scaled data

Starting Confusion Matrix (Scenario 1)



Final Confusion Matrix (Scenario 9)

```
File /opt/anaconda3/lib/python3.12/site-packages/pandas/core/dtypes/astype.py:101, in _astype_nansafe(arr, dtype, copy, skipna)
     96     return lib.ensure_string_array(
     97         arr, skipna=skipna, convert_na_value=False
     98     ).reshape(shape)
    100 elif np.issubdtype(arr.dtype, np.floating) and dtype.kind in "iu":
--> 101     return _astype_float_to_int_nansafe(arr, dtype, copy)
    103 elif arr.dtype == object:
    104     # if we have a datetime/timedelta array of objects
    105     # then coerce to datetime64[ns] and use DatetimeArray.astype
    107     if lib.is_np_dtype(dtype, "M"):

File /opt/anaconda3/lib/python3.12/site-packages/pandas/core/dtypes/astype.py:145, in _astype_float_to_int_nansafe(values, dtype, copy)
    141 """
    142 astype with a check preventing converting NaN to an meaningless integer value.
    143 """
    144 if not np.isfinite(values).all():
--> 145     raise IntCastingNaNError(
    146         "Cannot convert non-finite values (NA or inf) to integer"
    147     )
    148 if dtype.kind == "u":
    149     # GH#45151
    150     if not (values >= 0).all():

IntCastingNaNError: Cannot convert non-finite values (NA or inf) to integer
```

Note: Jupyter runs an error. The y-predict have many NaNs, which cannot be converted to integers. Cannot create a confusion matrix.

Conclusions

RNN with Unscaled Data

- Accuracy never goes beyond 0.644 while loss never drops below 10.00 (improved loss at expense of poor accuracy).
- Confusion Matrix recognizes the 15 stations.
- Confusion Matrix shows high inaccuracies for predicting pleasant weather (both true positive and false positive).

Comparing RNN Scaled vs Unscaled Data

- Scaled Data provides slightly higher accuracy and slightly lower loss.
- Confusion Matrix still shows high number of inaccuracies

CNN with Scaled Data

- Accuracy never reaches beyond 0.644.
- Lower loss but turns into NaN (not sure what this indicates)
- The NaN results block possibility for creating a confusion matrix.

Best choice

The RNN model with Scaled Data may be the best option in this part of the analysis, but the accuracy is far below that of the ANN model.