2.2: Complex Machine Learning Models and Keras Part 1 December 26, 2024
Isaac Contreras

CNN (Convolution Neural Network) model

1. Model Choice and Explanation

A CNN is chosen because it can handle local patterns in steps efficiently and trains faster than many RNN variants. I consider it a good baseline before exploring RNN or LSTM.

2. Hyperparameters

Keras Model

Hyperparameters

Initial Hyperparameters:

- Epochs = 10
- Batch size = 32
- n_hidden (filters for Conv1D or hidden units for RNN) = 64
- Activation (hidden layers) = "relu"
- Final layer activation = "softmax"
- Optimizer = "adam"
- Loss function = "categorical_crossentropy"

Possible Changes:

- Might increase `n_hidden` to 128 or 256 if accuracy is too low.
- Might increase `epochs` to 20 or 30 for better training.

```
# Basic hyperparameters
epochs = 10        # start small for testing
batch_size = 32
n_hidden = 64      # or 128, 256, etc.

timesteps = X_train.shape[1]   # 15
input_dim = X_train.shape[2]   # 9
n_classes = y_train.shape[1]   # 15
```

```
model = Sequential()
# Example: CNN with kernel_size=2
model.add(Conv1D(filters=n_hidden, kernel_size=2, activation='relu',
                 input_shape=(timesteps, input_dim)))
model.add(MaxPooling1D(pool_size=2))

model.add(Flatten())
model.add(Dense(32, activation='relu'))          # a small Dense layer
model.add(Dense(n_classes, activation='softmax'))
# Alternative final activations if multi-label or different constraints:
#  - 'sigmoid'  (common in multi-label classification)
#  - 'tanh'
#  - 'softmax'  (common in multi-class classification)

model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 14, 64)	1,216
max_pooling1d (MaxPooling1D)	(None, 7, 64)	0
flatten (Flatten)	(None, 448)	0
dense (Dense)	(None, 32)	14,368
dense_1 (Dense)	(None, 15)	495

Total params: 16,079 (62.81 KB)

Trainable params: 16,079 (62.81 KB)

Non-trainable params: 0 (0.00 B)

```python
# Sklearn confusion matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test_int, y_pred_int)
print("Confusion Matrix:\n", cm)

# Station names
stations = {
    0: 'BASEL', 1: 'BELGRADE', 2: 'BUDAPEST', 3: 'DEBILT',
    4: 'DUSSELDORF', 5: 'HEATHROW', 6: 'KASSEL', 7: 'LJUBLJANA',
    8: 'MAASTRICHT', 9: 'MADRID', 10: 'MUNCHENB', 11: 'OSLO',
    12: 'SONNBLICK', 13: 'STOCKHOLM', 14: 'VALENTIA'
}

y_test_names = pd.Series([stations[i] for i in y_test_int], name='True')
y_pred_names = pd.Series([stations[i] for i in y_pred_int], name='Pred')
ctab = pd.crosstab(y_test_names, y_pred_names)
print("\nCrosstab:\n", ctab)

# 3) MEASURE FINAL ACCURACY
accuracy = accuracy_score(y_test_int, y_pred_int)
print("\nFinal Accuracy (sklearn):", accuracy)
```

```
180/180 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
Confusion Matrix:
 [[   0 2599    0    0  550    0    0    0    0    0  424    0    0  109]
 [   0 1091    0    0    1    0    0    0    0    0    0    0    0    0]
 [   0  214    0    0    0    0    0    0    0    0    0    0    0    0]
 [   0   82    0    0    0    0    0    0    0    0    0    0    0    0]
 [   0   29    0    0    0    0    0    0    0    0    0    0    0    0]
 [   0   81    0    0    1    0    0    0    0    0    0    0    0    0]
 [   0   11    0    0    0    0    0    0    0    0    0    0    0    0]
 [   0   61    0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    9    0    0    0    0    0    0    0    0    0    0    0    0]
 [   0  401    0    0   52    0    0    0    0    0    4    0    0    1]
 [   0    8    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    4    0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    1    0    0    0    0    0    0    0    0    0    0    0    0]]

Crosstab:
 Pred        BELGRADE  DUSSELDORF  MUNCHENB  VALENTIA
True
BASEL           2599         550       424       109
BELGRADE        1091           1         0         0
BUDAPEST         214           0         0         0
DEBILT            82           0         0         0
DUSSELDORF        29           0         0         0
HEATHROW          81           1         0         0
KASSEL            11           0         0         0
LJUBLJANA         61           0         0         0
MAASTRICHT         9           0         0         0
MADRID           401          52         4         1
MUNCHENB           8           0         0         0
OSLO               5           0         0         0
STOCKHOLM          4           0         0         0
VALENTIA           1           0         0         0

Final Accuracy (sklearn): 0.19013593586615546
```

CNN Trials & Observations

1. **Initial Run**
    - **Hyperparameters**:
        - n_hidden = 64, epochs = 10, batch_size = 32
        - Activation (Conv layer) = **"relu"**, final layer = **"softmax"**
    - **Loss**: Ranged from ~1276 at epoch 1 to ~81105 by epoch 10. (It grew quite large over epochs.)
    - **Accuracy**: Reached around **19%** on the test set.
    - **Stations recognized**: Roughly **4** had non-zero diagonal in the confusion matrix (BELGRADE, DUSSELDORF, MUNCHENB, VALENTIA).
    - **Observation**: The model mostly predicts a few classes (huge confusion-matrix counts in those columns).
2. **Increase n_hidden to 128, epochs = 20**
    - **Hyperparameters**:
        - n_hidden = 128, epochs = 20, batch_size = 32
        - Activation (hidden) = **"relu"**, final = **"softmax"**
    - **Loss**: Still grows quite large (reaches the 100k+ range, sometimes 200k+).
    - **Accuracy**: ~**19%** in one run, or ~**9.5%** in another, depending on the specific run.
    - **Stations recognized**: Usually sees a heavy skew toward “MADRID” or “BELGRADE,” with many stations never predicted.
    - **Observation**: Doubling the hidden units did not significantly improve recognition of all stations—some confusion matrices show even **lower** accuracy.
3. **Increase n_hidden to 260, epochs = 20**
    - **Hyperparameters**:
        - n_hidden = 260, epochs = 20, batch_size = 32
        - Activation = **"relu"**, final = **"softmax"**
    - **Loss**: Continues to escalate into the 100k+ or 600k+ range.
    - **Accuracy**: Dropped to about **7.9–9%** in the final runs.
    - **Stations recognized**: The confusion matrix is dominated by one predicted class (e.g., “MADRID”), with many rows having zero correct predictions.

    - **Observation**: The bigger filter count alone didn't help; the model remains heavily biased, possibly due to unscaled inputs or class imbalance.

Conclusions / Next Steps

- **Softmax** activation is used for the final layer throughout, which is correct for multi-class classification. However, large losses suggest the network struggles.
- **Increasing filters** (n_hidden from 64 → 128 → 260) *did not* straightforwardly improve accuracy.
- **Accuracy** ranges from about **19%** at best to **7–9%** at worst. Most predictions go to a single station (e.g. BELGRADE or MADRID).
- **Potential reason**: data may need **scaling** or more **epochs**, or you might need **class weighting** for imbalanced data.
- **Stations recognized**: No run so far fully recognizes all 15 stations; often only a handful appear on the diagonal.