2.2: Complex Machine Learning Models and Keras Part 1

Innocent Bayai

9 December 2024 (Revised)

Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) models are typically used for spatially structured data such as images, videos, and even 2D or 3D data where local features are important. CNNs use convolutions to capture hierarchical patterns in spatial data. For example, CNN models can be used in image classification, and they can automatically learn to recognize edges, shapes, and objects in images. CNNs are also compatible with high dimensional numerical data – making the model ideal for efficient learning of weather patterns that are spatially distributed.

I run the CNN model several times (1st round to 3rd round) using different hyperparameters to get to a convergence where accuracy is maximized whilst losses are minimized. The table hereunder summarizes all iterations made.

NB: Not all estimates made are presented in the table. I presented those deemed to have material effect on the decision made at the end.

1st Round Estimations

	1st run	*2nd run*	*3rd run*	*4th run*
Epochs	*30*	*60*	*50*	*50*
Batch_size	*32*	*64*	*64*	*450*
n_hidden	*256*	*512*	*512*	*1000*
Activation	*relu, softmax*	*relu, softmax*	*relu, softmax*	*Relu, softmax*
Accuracy	*12.80%*	*11.69%*	*11.11%*	*13.52%*
Loss	*31537466*	*51384124*	*32790606*	*537278*

After this stage, I considered the estimation hyperparameters under the 4th run and tried different combinations of activation methods (relu, tank, sigmoid) whilst also altering the hyperparameters from the 4th run.

2nd Round of Estimations (activation method remained relu and softmax)

For this section, I shorten the presentation for epochs, batch_size and n_hidden as follows @50;450;1000. This represents 50 epochs, 450 batch_size, and 1000 n_hidden. This allowed me to shorten my iterations. My findings are as follows:

i. *Considering @ 50;450;2000 reduced accuracy to 8.06% and a loss of 792491. I therefore decide to retain 1000 n_hidden.*

ii. *I try @50;500;1000 and accuracy increase to 15.12% and losses are at 910930.*

iii. *I run @50;550;1000 and accuracy is further reduced, thus I decided to assume @50;500;1000.*

3rd Round Estimations (using different combinations of relu, tanh, softmax, and sigmoid)

After retaining ii. from the second round as the best estimate, I then play around with different activation methods.

i. *Using relu, relu, tanh and @50;550;1000 gives the accuracy of 63.14, and a loss of 26.8490*

ii. *Using relu, relu, tanh and @60;550;1000 reduces accuracy to 19.06%, so I retain 50 epochs.*

iii. *Using relu, relu, tanh and @ 40;550;1000 gives accuracy of 59.51%, I retain 50 epochs.*

iv. *Using relu, relu, tanh and @50;560;1000 reduces accuracy to 8.96%, so I retain 550 batch size.*

v. *Using relu, relu, tanh and @50;500;1000 gives accuracy of 29.78 so I retain 550 batch size.*

vi. *Using relu, relu, tanh and @50;550;1100 reduces accuracy to 3.49%, I retain @50;550;1000*

vii. *Using relu, relu, tanh and @50;550;900 reduces accuracy to 1.54%, I retain @50;550;1000.*

viii. *Using relu, relu, relu@50;550;1000 gives accuracy of 64.35, the highest with nan losses (see the pictured appended underneath note below.*

NB: I tried different activation methods using Sigmoid and softmax but they all give lower accuracy. The best results are shown below.

```
epochs = 50
batch_size = 550
n_hidden = 1000

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

```
Epoch 1/50
26/26 - 3s - 99ms/step - accuracy: 0.0720 - loss: 20.1095
Epoch 2/50
26/26 - 1s - 40ms/step - accuracy: 0.0622 - loss: 19.3343
Epoch 3/50
26/26 - 1s - 39ms/step - accuracy: 0.0749 - loss: 19.2975
Epoch 4/50
26/26 - 1s - 39ms/step - accuracy: 0.0748 - loss: 19.2551
Epoch 5/50
26/26 - 1s - 40ms/step - accuracy: 0.0813 - loss: 19.2757
Epoch 6/50
26/26 - 1s - 40ms/step - accuracy: 0.0858 - loss: 19.1965
Epoch 7/50
26/26 - 1s - 39ms/step - accuracy: 0.0765 - loss: 19.1511
Epoch 8/50
26/26 - 1s - 40ms/step - accuracy: 0.0862 - loss: 19.1254
Epoch 9/50
26/26 - 1s - 40ms/step - accuracy: 0.1060 - loss: 19.1457
Epoch 10/50
26/26 - 1s - 40ms/step - accuracy: 0.1386 - loss: 19.1226
Epoch 11/50
26/26 - 1s - 39ms/step - accuracy: 0.1155 - loss: 19.0621
Epoch 12/50
26/26 - 1s - 39ms/step - accuracy: 0.1192 - loss: 19.0395
Epoch 13/50
26/26 - 1s - 41ms/step - accuracy: 0.1373 - loss: 18.9973
Epoch 14/50
26/26 - 1s - 39ms/step - accuracy: 0.1584 - loss: 19.0002
Epoch 15/50
26/26 - 1s - 38ms/step - accuracy: 0.1837 - loss: 19.2477
Epoch 16/50
26/26 - 1s - 44ms/step - accuracy: 0.1341 - loss: 17.4291
Epoch 17/50
26/26 - 1s - 38ms/step - accuracy: 0.0694 - loss: 16.1631
Epoch 18/50
26/26 - 1s - 40ms/step - accuracy: 0.0817 - loss: 16.0916
Epoch 19/50
```

```
Epoch 19/50
26/26 - 1s - 39ms/step - accuracy: 0.1065 - loss: 16.0044
Epoch 20/50
26/26 - 1s - 39ms/step - accuracy: 0.1862 - loss: 15.5874
Epoch 21/50
26/26 - 1s - 42ms/step - accuracy: 0.1124 - loss: 15.4573
Epoch 22/50
26/26 - 1s - 42ms/step - accuracy: 0.1105 - loss: 15.3466
Epoch 23/50
26/26 - 1s - 39ms/step - accuracy: 0.1387 - loss: 15.2959
Epoch 24/50
26/26 - 1s - 40ms/step - accuracy: 0.1198 - loss: 15.4487
Epoch 25/50
26/26 - 1s - 42ms/step - accuracy: 0.1501 - loss: 15.3736
Epoch 26/50
26/26 - 1s - 39ms/step - accuracy: 0.1575 - loss: 15.2598
Epoch 27/50
26/26 - 1s - 40ms/step - accuracy: 0.1686 - loss: 15.2523
Epoch 28/50
26/26 - 1s - 40ms/step - accuracy: 0.1337 - loss: 15.2478
Epoch 29/50
26/26 - 1s - 41ms/step - accuracy: 0.1790 - loss: 15.2914
Epoch 30/50
26/26 - 1s - 39ms/step - accuracy: 0.1641 - loss: 14.3757
Epoch 31/50
26/26 - 1s - 39ms/step - accuracy: 0.1492 - loss: 12.4925
Epoch 32/50
26/26 - 1s - 39ms/step - accuracy: 0.1654 - loss: 12.4469
Epoch 33/50
26/26 - 1s - 38ms/step - accuracy: 0.1828 - loss: 12.4029
Epoch 34/50
26/26 - 1s - 39ms/step - accuracy: 0.1858 - loss: 12.3959
Epoch 35/50
26/26 - 1s - 39ms/step - accuracy: 0.1819 - loss: 12.3784
Epoch 36/50
26/26 - 1s - 40ms/step - accuracy: 0.0792 - loss: 12.6284
Epoch 37/50
26/26 - 1s - 40ms/step - accuracy: 0.1336 - loss: 12.5482
```

```
Epoch 38/50
26/26 - 1s - 39ms/step - accuracy: 0.1698 - loss: 12.4661
Epoch 39/50
26/26 - 1s - 40ms/step - accuracy: 0.1810 - loss: 12.4280
Epoch 40/50
26/26 - 1s - 45ms/step - accuracy: 0.1869 - loss: 12.3875
Epoch 41/50
26/26 - 1s - 40ms/step - accuracy: 0.1884 - loss: 12.4235
Epoch 42/50
26/26 - 1s - 41ms/step - accuracy: 0.1464 - loss: 12.6153
Epoch 43/50
26/26 - 1s - 40ms/step - accuracy: 0.1885 - loss: 12.5120
Epoch 44/50
26/26 - 1s - 41ms/step - accuracy: 0.1846 - loss: 12.3891
Epoch 45/50
26/26 - 1s - 39ms/step - accuracy: 0.1920 - loss: 12.3849
Epoch 46/50
26/26 - 1s - 40ms/step - accuracy: 0.1919 - loss: 12.3393
Epoch 47/50
26/26 - 1s - 41ms/step - accuracy: 0.4309 - loss: nan
Epoch 48/50
26/26 - 1s - 42ms/step - accuracy: 0.6435 - loss: nan
Epoch 49/50
26/26 - 1s - 40ms/step - accuracy: 0.6435 - loss: nan
Epoch 50/50
26/26 - 1s - 42ms/step - accuracy: 0.6435 - loss: nan
]: <keras.src.callbacks.history.History at 0x27492d54ad0>
```

Based on the best accuracy estimate, the confusion matrix hereunder suffices.

```
[331]: # Evaluate

       print(confusion_matrix(y_test, model.predict(X_test)))

       180/180 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

Pred	BASEL
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

Recurrent Neural Network (RNN)

The RNN model used less epochs, lower batch size, and less n_hidden compared to the CNN model. The only model run assumes 30 epochs, batch size 16, and n_hidden of 64. The model also used relu and tanh activation methods as shown the picture below. I did

few iterations and settled for the @30;16;64 as the accuracy almost matched the CNN and the loss was minimal too at 24.52%. Later to be discovered, the confusion matrices of the two methods are the same.

```
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(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 RNN model registered the following results:

```
Epoch 1/30
1076/1076 - 14s - 13ms/step - accuracy: 0.0981 - loss: 24.5115
Epoch 2/30
1076/1076 - 8s - 8ms/step - accuracy: 0.1003 - loss: 24.7179
Epoch 3/30
1076/1076 - 8s - 7ms/step - accuracy: 0.1282 - loss: 25.0474
Epoch 4/30
1076/1076 - 8s - 8ms/step - accuracy: 0.1110 - loss: 24.5681
Epoch 5/30
1076/1076 - 8s - 7ms/step - accuracy: 0.0766 - loss: 24.9231
Epoch 6/30
1076/1076 - 8s - 8ms/step - accuracy: 0.0942 - loss: 25.0130
Epoch 7/30
1076/1076 - 8s - 7ms/step - accuracy: 0.0636 - loss: 24.3672
Epoch 8/30
1076/1076 - 8s - 8ms/step - accuracy: 0.0217 - loss: 24.3082
Epoch 9/30
1076/1076 - 8s - 7ms/step - accuracy: 0.0221 - loss: 24.3728
Epoch 10/30
1076/1076 - 8s - 8ms/step - accuracy: 0.0268 - loss: 24.5799
Epoch 11/30
1076/1076 - 8s - 7ms/step - accuracy: 0.0388 - loss: 24.3845
Epoch 12/30
1076/1076 - 8s - 8ms/step - accuracy: 0.0265 - loss: 24.5429
Epoch 13/30
1076/1076 - 8s - 7ms/step - accuracy: 0.0130 - loss: 24.7430
Epoch 14/30
1076/1076 - 8s - 8ms/step - accuracy: 0.0605 - loss: 24.5607
Epoch 15/30
1076/1076 - 8s - 7ms/step - accuracy: 0.0769 - loss: 24.5942
Epoch 16/30
1076/1076 - 8s - 8ms/step - accuracy: 0.0816 - loss: 24.4778
Epoch 17/30
1076/1076 - 8s - 7ms/step - accuracy: 0.0826 - loss: 24.4086
Epoch 18/30
1076/1076 - 8s - 7ms/step - accuracy: 0.1113 - loss: 24.4749
Epoch 19/30
1076/1076 - 8s - 7ms/step - accuracy: 0.0574 - loss: 24.2849
```

```
Epoch 17/30
1076/1076 - 8s - 7ms/step - accuracy: 0.0826 - loss: 24.4086
Epoch 18/30
1076/1076 - 8s - 7ms/step - accuracy: 0.1113 - loss: 24.4749
Epoch 19/30
1076/1076 - 8s - 7ms/step - accuracy: 0.0574 - loss: 24.2849
Epoch 20/30
1076/1076 - 8s - 8ms/step - accuracy: 0.0766 - loss: 24.4173
Epoch 21/30
1076/1076 - 8s - 7ms/step - accuracy: 0.1544 - loss: 24.6660
Epoch 22/30
1076/1076 - 8s - 8ms/step - accuracy: 0.2847 - loss: 25.1254
Epoch 23/30
1076/1076 - 8s - 8ms/step - accuracy: 0.2521 - loss: 24.7906
Epoch 24/30
1076/1076 - 8s - 8ms/step - accuracy: 0.3584 - loss: 24.4729
Epoch 25/30
1076/1076 - 8s - 7ms/step - accuracy: 0.4399 - loss: 24.5224
Epoch 26/30
1076/1076 - 8s - 8ms/step - accuracy: 0.5083 - loss: 24.5773
Epoch 27/30
1076/1076 - 8s - 7ms/step - accuracy: 0.5475 - loss: 24.6390
Epoch 28/30
1076/1076 - 9s - 8ms/step - accuracy: 0.5332 - loss: 24.4036
Epoch 29/30
1076/1076 - 8s - 8ms/step - accuracy: 0.5790 - loss: 24.6467
Epoch 30/30
1076/1076 - 8s - 8ms/step - accuracy: 0.6213 - loss: 24.5205
[56]: <keras.src.callbacks.history.History at 0x2df8b4c9370>
```

The resultant confusion matrix for the RNN model is shown below. Its values are the same as those for the CNN model.

```
180/180 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step
Pred         BASEL
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
```

Comments:

Accuracy: Initially very low (~0.0981), but it steadily improves during the training process. By epoch 30, the model achieves an accuracy of 0.6213, indicating some progress in learning. The accuracy level is almost the same as that for CNN, standing at 64.35%.

Loss: The loss value starts at 24.5115 and fluctuates during the epochs but doesn't increase as significantly as in the first model. It ends at 24.5205 in epoch 30, indicating that the model's loss is relatively stable or only slightly decreasing.

Training duration: Each epoch takes 8 seconds, and the model processes 1076 steps per epoch. The time per step is relatively fast at 8ms per step.

Comparison – CNN vs RNN: The CNN model had a training time of 1 second per epoch (with 538 steps per epoch), which is significantly faster than the RNN model where each epoch took 8 seconds. The CNN model is likely simpler or has fewer parameters, making it faster to train.

The **accuracy** for the CNN model was higher, ending at 64.35%, while the RNN model is slightly below, reaching at 62.13% by epoch 30.

Loss Stability: The RNN model exhibits better loss stability at 24 compared to the CNN model. However, the CNN managed to end with loss of nan by epoch 30.

The RNN Model produced the confusion matrix showing hereunder. Interestingly, both the CNN and the RNN are giving the same confusion matrix implying that the two models are seemingly equally good.

END