Classifier for multivariate time series classification using RNN and CNN

[Author: Anustup Das]

DataSet:

The provided data set is 'NATOPS' multivariate time series dataset. This Dataset have equal instances of Training and Test Data where there are 180 time series each have 51 timesteps and the number of features for each timestep is 24. Each instance is associated with a target label which have 6 different class values ranging from "1" to "6"

Approach:

The task is to design two classifiers using CNN and RNN and compare the results.

For the RNN model Long Short-Term Memory (LSTM) recurrent neural networks was chosen as it offer lot of flexibility in modelling the problem — meaning we have a good control over several parameters of the time series.

For CNN a 1D CNN model have been designed for the classification task.

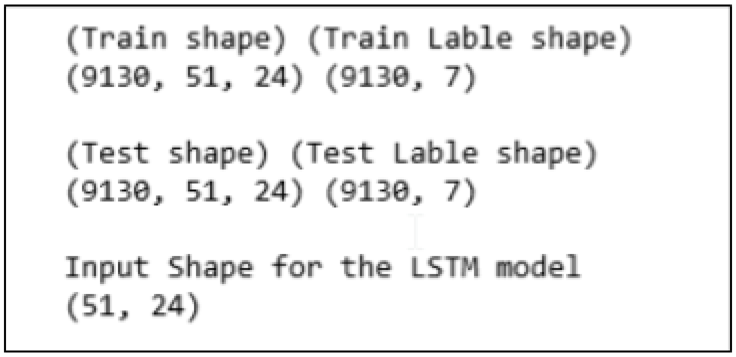
Data - Preparation:

Both LSTM and CNN expects the time-series data to be in a shape of a 3D tensor with the following structure:
['Sample_number', 'TimeStep_number','Feature_number'].
The data was loaded into a pandas dataframe and the features and labels were segregated respectively in X and Y *[train_data and train_target].*

Initially the shape of the X was *['Sample_number','Feature_number']* and Y was *['Sample_number',1]* as the label comprised of only one column.
The labels were transformed into "**onehot encoding**" form for easier interpretation for the LSTM and CNN models.

X was reshaped into an numpy array of the total data divided into several time series with 51 instances each as the number of timesteps. Each time step has 24 feature input. So the reshaped data has the from of:

['Sample_number','timestep_number','Feature_number'] e.g [9130,51,24]. And the shape of the lable data is now *['Sample_number',7].*



```
(Train shape) (Train Lable shape)
(9130, 51, 24) (9130, 7)

(Test shape) (Test Lable shape)
(9130, 51, 24) (9130, 7)

Input Shape for the LSTM model
(51, 24)
```

LSTM Classification Model:

We define the LSTM model with the following specifications along with different values tried for tuning the hyperparameters:

Input Shape : 51 time step 24 features.

1st Hidden Layer: LSTM [32/51/64 neurons]

2nd Hidden Layer: Dropout [0.5/0.2 fraction of the input units to drop] *//To reduce overfitting.*

3rd Hidden Layer: Dense [100 neurons, Activation function : relu/tanh]

Output Layer: [7 neurons, Activation function : softmax] *//For predicting class in onehot encoded form*

Loss Function: 'categorical_crossentropy'

Optimizer: Adam/SDG/Adamax

Learning Rate: .01/.05/.001/.004

Number of Epochs: 100

Batch Size: 51/64/100

The model trains for 100 epoch with a callback function which keeps track of both accuracy and loss and when we get a global minimum loss from the model the weights of that particular epoch is stored in a file named ***"LSTM_Best-weights-my_model-{epoch_no}-{loss}-{accuracy}.hdf5"*** along with a stopping criteria which will monitor the training for n number of iterations and if the global minimum loss is not improved within that n iteration the training will be stopped.

Note: The final parameters which yielded best results the present in the submitted code.

Evaluating Model:

In this experiment we allowed the model to train for the total number of epochs without any stopping criteria. After training we plot the training and validation accuracy along with the loss over the total number of epochs and analysis where the model performed best. We then model the model with the saved weight of the best epochs.

The log for every epoch is stored in the file : "*MultiVariate_LSTM_model_train.csv*"

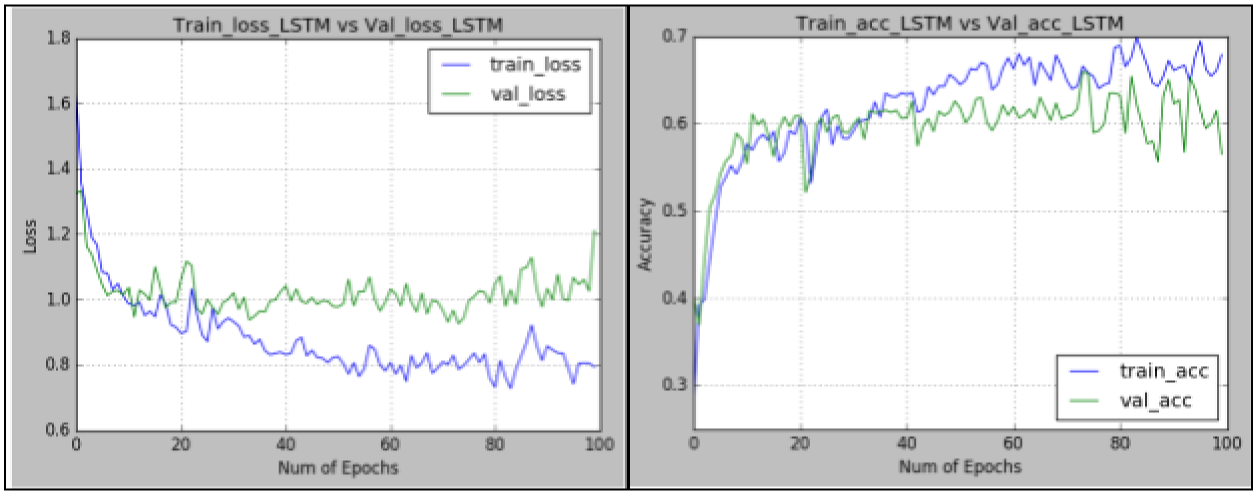


Fig: Graph showing the loss and accuracy over train and validation data for 100 epochs.

The model in this experiment achieved its lowest loss value as **0.92641** and highest accuracy value as **0.6596.**

CNN Classification Model:

We define the CNN model with the following specifications along with different values tried for tuning the hyperparameters:

Input Shape : 51 time step 24 features.

1st Hidden Layer: Conv1D [32/51/64 neurons, Activation function : relu]

2nd Hidden Layer: MaxPooling1D [pool_size : 2/3]

3rd Hidden Layer: Flatten

4th Hidden Layer: Dropout [0.5/0.2 fraction of the input units to drop] *//To reduce overfitting.*

5th Hidden Layer: Dense [50/64/100 neurons, Activation function : relu/tanh]

Output Layer: [7 neurons, Activation function : softmax] *//For predicting class in onehot encoded form*

Loss Function: 'categorical_crossentropy'

Optimizer: Adam/SDG/Adamax

Learning Rate: .01/.05/.001/.004

Number of Epochs: 100

Batch Size: 51/64/100

The model trains for 100 epoch with a callback function which keeps track of both accuracy and loss and when we get a global minimum loss from the model the weights of that particular epoch is stored in a file named ***"CNN_Best-weights-my_model-{epoch_no}-{loss}-{accuracy}.hdf5"*** along with a stopping criteria which will monitor the training for n number of iterations and if the global minimum loss is not improved within that n iteration the training will be stopped.

Note: The final parameters which yielded best results the present in the submitted code.

Evaluating Model:

In this experiment we allowed the model to train for the total number of epochs without any stopping criteria. After training we plot the training and validation accuracy along with the loss over the total number of epochs and analysis where the model performed best. We then model the model with the saved weight of the best epochs.

The log for every epoch is stored in the file : “*MultiVariate_CNN_model_train.csv*”

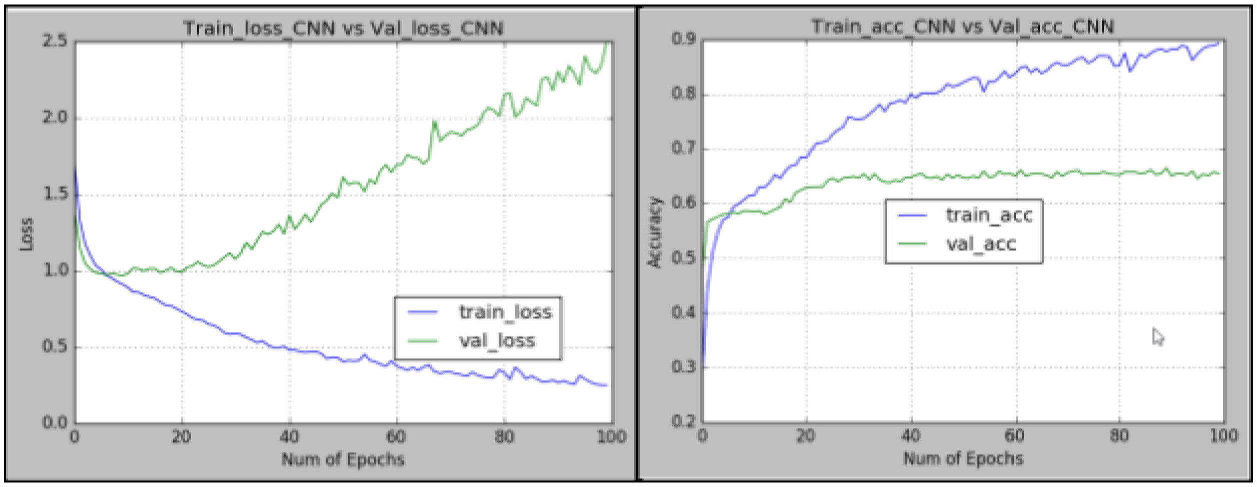


Fig: Graph showing the loss and accuracy over train and validation data for 100 epochs.

The model in this experiment achieved its lowest loss value as **0.96616** and accuracy value as **0.6539.**

Result Comparison:

The Validation loss and Validation accuracy of both the LSTM and CNN model were compared and the below figure displays the results of the experiment.

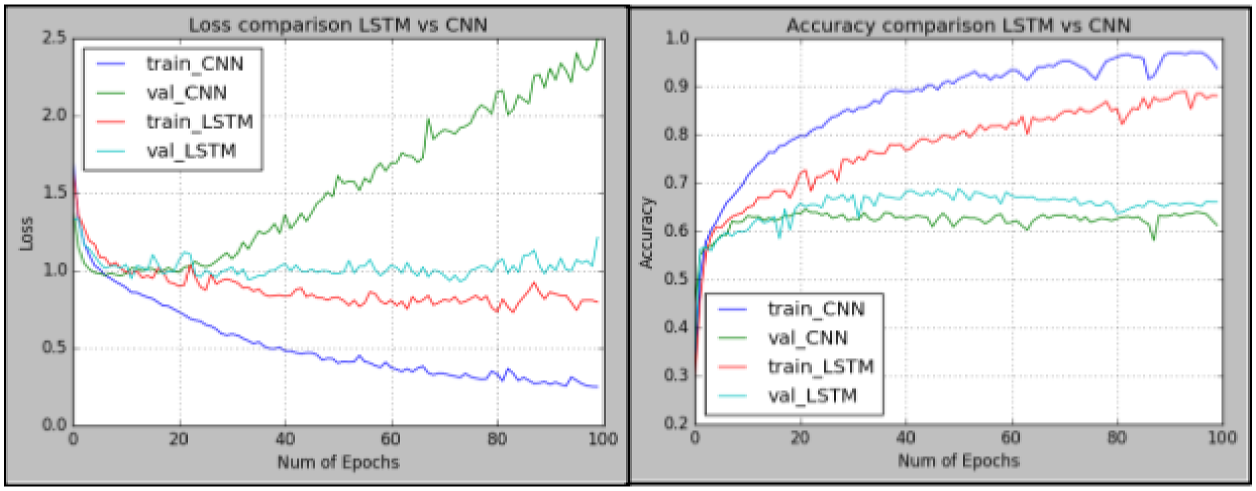
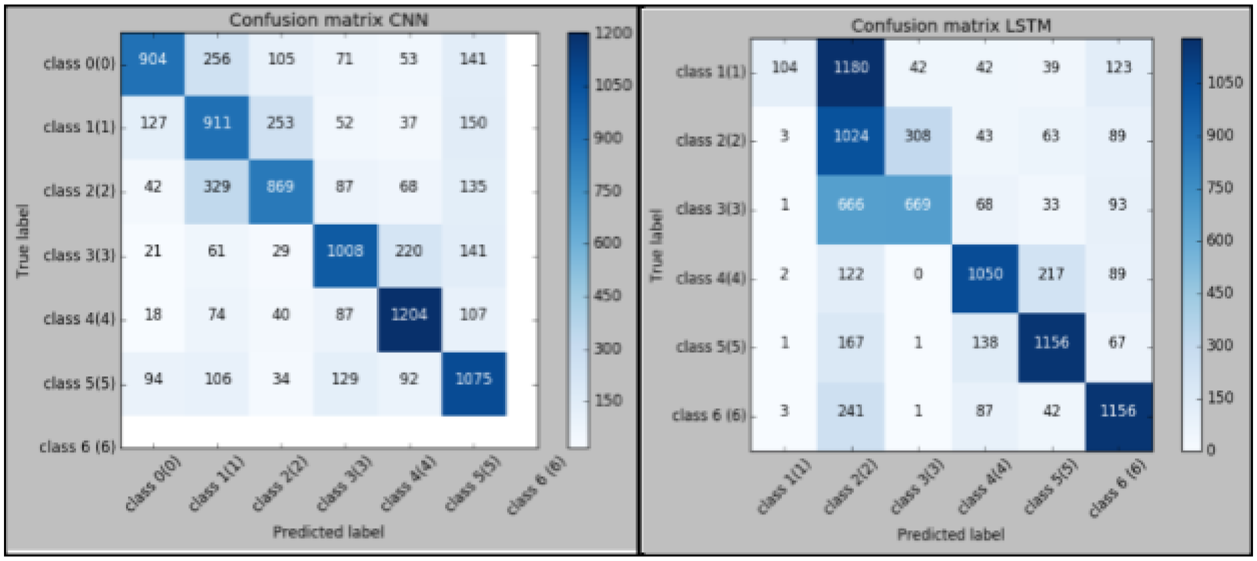


Fig: Graph showing comparison between LSTM and CNN ove loss and accuracy on train and validation data for 100 epochs.

As seeing from the above graph the LSTM classifier appears to perform better than the CNN classifier both in terms of low validation loss and high validation accuracy. It also seems that the CNN model overfits the training data more than LSTM. But while comparing the confusion matrix of both the model CNN shows better classification result than the LSTM solution where the results are not diagonally distributed over the confusion matrix in comparison to the CNN model.



Confusion matrix CNN

	class 0(0)	class 1(1)	class 2(2)	class 3(3)	class 4(4)	class 5(5)	class 6 (6)
class 0(0)	904	256	105	71	53	141	
class 1(1)	127	911	253	52	37	150	
class 2(2)	42	329	869	87	68	135	
class 3(3)	21	61	29	1008	220	141	
class 4(4)	18	74	40	87	1204	107	
class 5(5)	94	106	34	129	92	1075	
class 6 (6)							

Confusion matrix LSTM

	class 1(1)	class 2(2)	class 3(3)	class 4(4)	class 5(5)	class 6 (6)
class 1(1)	104	1180	42	42	39	123
class 2(2)	3	1024	308	43	63	89
class 3(3)	1	666	669	68	33	93
class 4(4)	2	122	0	1050	217	89
class 5(5)	1	167	1	138	1156	67
class 6 (6)	3	241	1	87	42	1156

Fig: Confusion matrix showing comparison between CNN and LSTM.

Future Scope:

There is a lot of scope of improvement of the models with furthermore tuning of the hyper parameters and observing the effects on the model until we can achieve highest accuracy to a range of 90 to 95% with much lower loss.
Another model can be developed by concatenating both the LSTM and CNN model for better performance.