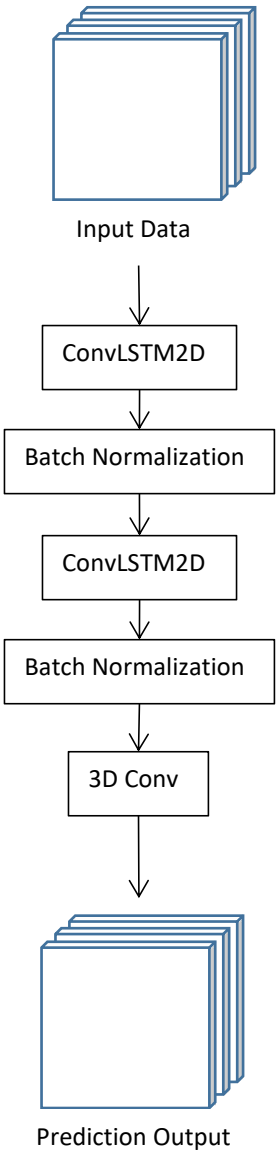
Grab Traffic Management Challenge

Summary

The grab traffic management challenge is a spatial temporal problem. Hence the approach taken here will be to use a convolutional LSTM to try and model the problem, convolution to capture the spatial information, and LSTM to capture the temporal information. The convolution filter will be using a kernel of size 3x3 on the assumption that only direct neighbours will have any influence on each of the geohash.

The given data was inspected and there was not much data cleaning to be done, since there is no invalid data. We only need to convert the raw data into matrix format suitable to feed into a 2D Convolutional LSTM network for training and fill in the missing data points with zeros.



The general Model adopted in this solution is as given on the left.

A range of model parameters were used to create different Models:

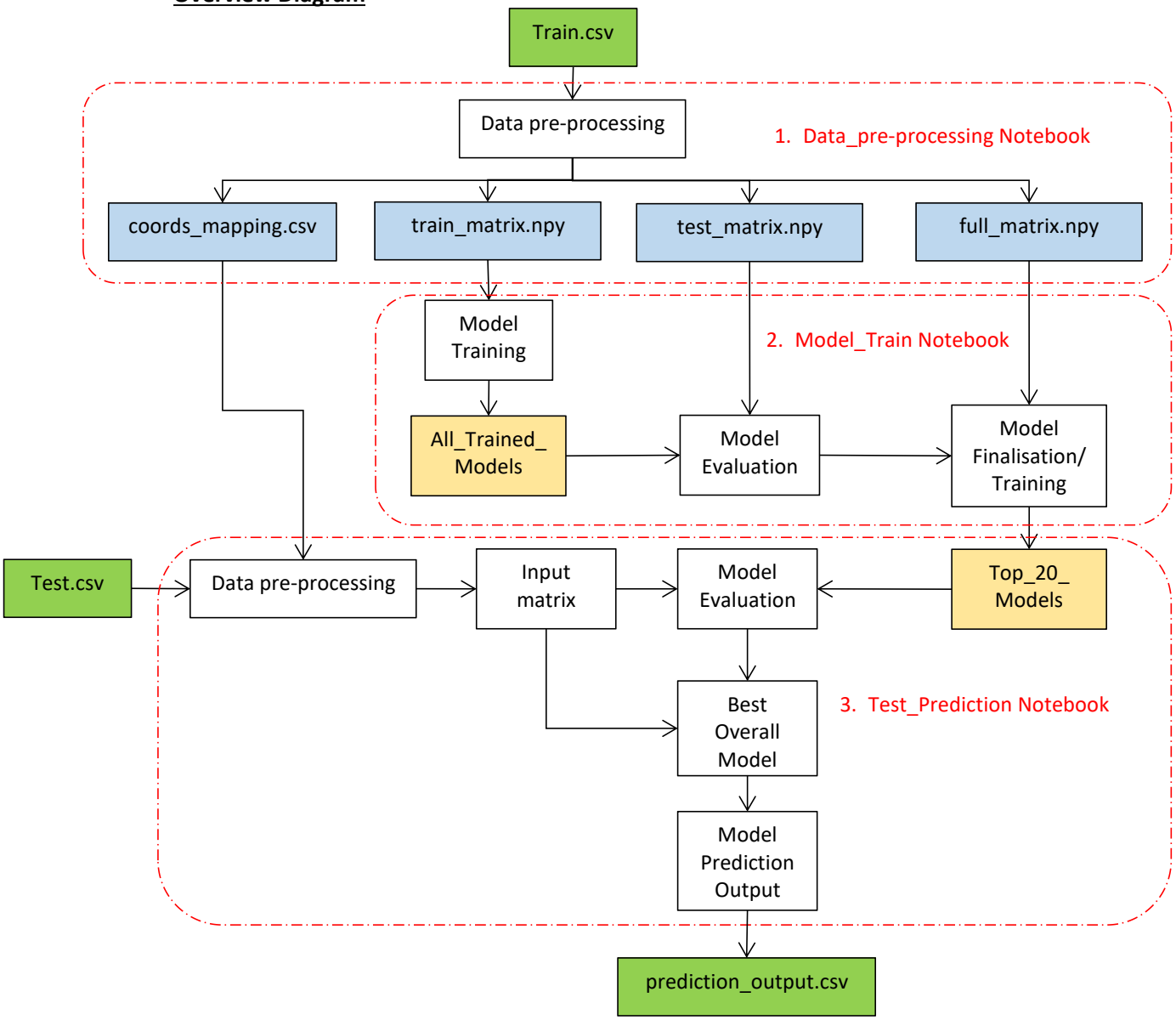
1. Filter layer (convolutional)
2. Number of LSTM Layers
3. Batch sequence training of x days
   (feeding in 1 day/ 2 day/ 14 day per sample)
4. Dropout Rate (convolutional)
5. Recurrent Dropout Rate (LSTM)

It was found that 40 convolutional filter layers, 2 LSTM layers, 1 day batch sequence, 0 or 0.1 dropout and 0.5 recurrent dropout gives the overall best result performance.

All the trained models were then evaluated on a hold-out test set and the top 20 models with the best RMSE score were then trained on the full training data set to prepare for the actual test prediction.The best model produced an average RMSE score of 0.02 on the hold-out test set.

During the actual test prediction, all 20 models will be evaluated again for RMSE score based on the test set, and the best model is used to output the final prediction.

Overview Diagram



The Diagram above gives an overview of the whole process flow.

The following pages will explain in detail what each of the notebook does.

1. Data_pre-processing Notebook

The raw data given for this challenge:

geohash6	day	timestamp	demand
qp03wc	18	20:00	0.020071791
…	…	…	…

The first step is to process the data into a suitable format to feed into a model.

This is a spatial-temporal problem, so we are going to process the raw data into spatial temporal form, and feed into a 2D Convolutional LSTM model for training.

Processing geohash6

Using geohash.decode_exactly function, we get back lat, long, lat_err and long_err for each geohash. Take note we need to use geohash.decode_exactly and not geohash.decode as it is not accurate enough.

geohash6	lat	long	lat_err	long_err
qp03wc	-5.353088	90.653687	0.002747	0.005493
…	…	…	…	…

Next, we then arrange the latitude and longitude in increasing order and index them, starting from 0. This is for easy looping and mapping to 2D later on.

lat_index	lat
0	-5.484924
1	-5.479431
…	…
43	-5.248718
44	-5.243225
45	-5.237732

long_index	long
0	90.587769
1	90.598755
…	…
33	90.950317
34	90.961304
35	90.97229

We then store the table above in order to perform the same mapping on the test data set.
The table is stored as ***coords_mapping.csv***

Processing day & timestamp

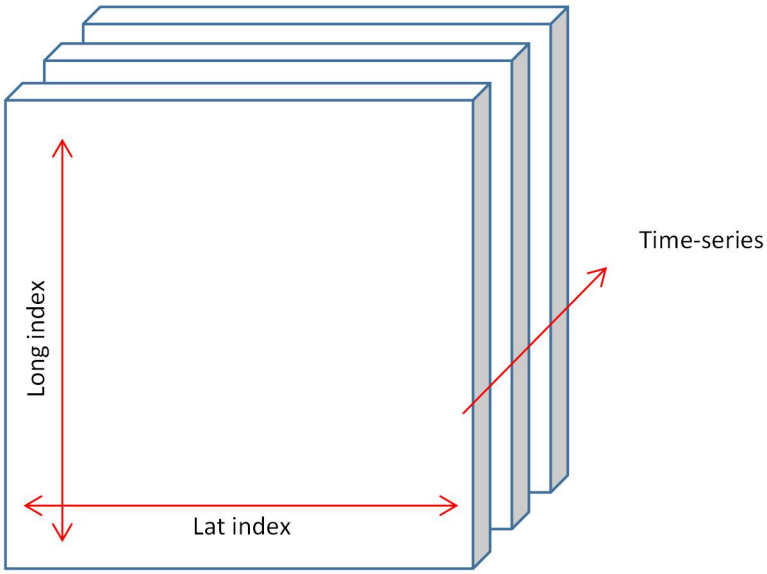
First, we convert the timestamp into an integer sequential format. The given raw data is in 15 min buckets, so there will be 96 time-steps in a day (4 x 24 = 96). There are a total of 61 days in the dataset, so the total time sequence will be (96 x 61 = 5856).

Processing into 2D Matrix

After we process the geohash and the day / time stamps, we get the table like below:

demand	lat_index	long_index	time-series
0.009146	13	0	0
0.011477	25	0	0
0.019114	14	0	0
…	…	…	…
0.003026	34	34	5855
0.002359	35	35	5855
0.018874	37	35	5855

We then map the data into 2D time series format as below:



The actual matrix is of shape (1,5856,36,46,1) because the ConvLSTM2D takes input of shape [n_samples, time-series, longtitude, lattitude, depth]. Here, our depth is =1.

We then split the data into train and test set at day 46 and save the data into .npy format for further processing.

full_matrix.npy - The full matrix for 61 days
train_matrix.npy - Matrix from day 1 to 45
test_matrix.npy - Matrix from day 46 to 61

2. Model_Train Notebook

Model Training

We defined a function *generate_batch* to generate input sets and target sets batches based on x days per batch. This is done in order to find out the optimal number of sequence to feed into the network for training per batch. It was found the optimal number to train a model is around 1 day per batch, training the LSTM model with more than 2 days per batch returns bad results.

Seq_Model function was defined to enable us to create models based on a set of parameters, ie filter size, number of layers, dropout rate and recurrent dropout rate.

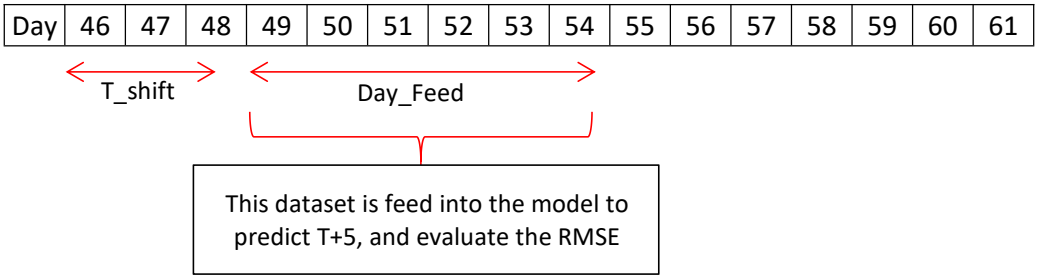
We then loop through and train for different parameters, and store all the trained models into the folder ***All_Trained_Models.*** 3 different files are saved per model, they are the model parameters, model weights and training history. The files are named according to the model parameters.

The different parameters altered are filter layers (convolutional), model layers, batch size, dropout rate and recurrent dropout rate.

Model Evaluation

We then load **test_matrix.npy (Matrix from day 46 to 61)** and evaluate the RMSE score of each model rigorously. We use two ways to shift data and get predictions before evaluating the RMSE score of the model, T_shift and Day_Feed. This is to ensure we can select a robust model which performs well across multiple time steps.

T_shift shifts the test data to the left by a number of time series steps
Day_Feed controls the total number of days to feed into the model for prediction



Baseline RMSE is the RMSE you get when using the given demand values at T to be values for (T+1 to T+5). This is the minimum RMSE to beat, as a model is considered useless if just blindly using the last seen values can outperform the model prediction.

Model Finalisation

After the model evaluation was done for all the trained models, the top 20 models are then selected based on best performing RMSE score to train on the full training dataset. The 20 trained models is then stored into the folder ***Top_20_Models***. During the actual test prediction, we will use and load the models from here.

3. Test_Prediction Notebook

The file ***test.csv*** is loaded and converted to the same form as the input matrix for the LSTM model. ***coords_mapping.csv*** is used so that coordinates mapping of the test dataset is the same as the coordinates mapping done during model training.

RMSE evaluation will be performed on the 20 selected models from ***Top_20_Models*** based on the test data set, and the top model from this evaluation selected for prediction on the test set.

There is no specification in the challenge of the required output format, so we reformat the prediction into the most easily readable format of the form:

geohash6	T+1	T+2	T+3	T+4	T+5
…	…	…	…	…	…

4. EDA & Model Visualization Notebook

The first part of this notebook explores the given data, and shows that there are no null values and all the target values are already normalised and within the 0 to 1 range.

The second part shows the model training history and visualize the model output, both in 2D format and in numerical format. This is to check whether the model is training well and identify the cause of the model errors.

Future Direction

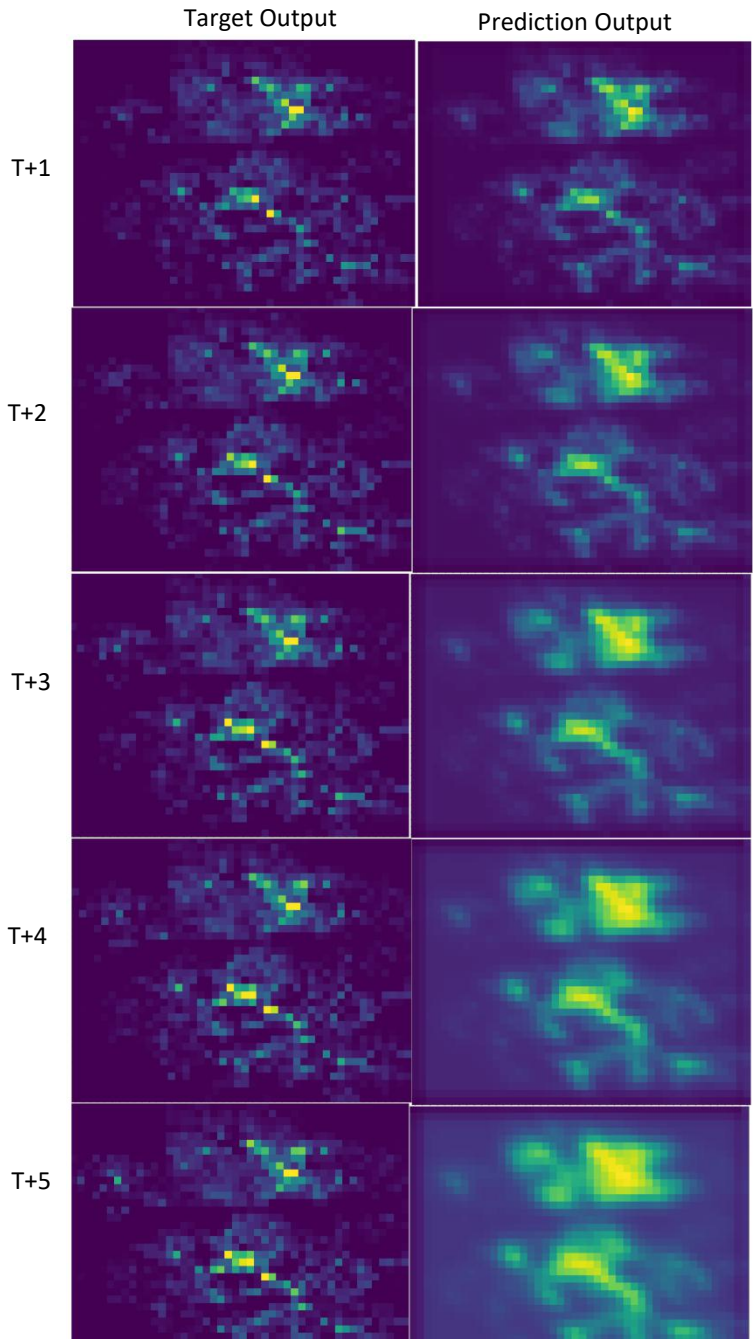
1. Try to perform LSTM model training on individual geohash
   - This is to test whether the spatial relations between geohashes are actually important
2. Implement cuDNN Conv LSTM for faster iteration and training
   - Could be used to speed up training and model iterations to explore more model parameters.
3. Initialize model weights
   - It seems like some of the models trained successfully and some diverges completely, could try to use some initializers to test whether it helps make model training more consistent.

Additional Notes

Initially was thinking of doing feature extraction and using LightGBM/XGBoost to fit the data on, but realized it would be tedious and would not necessarily be able to capture the spatial and temporal information. So I did some research and came across 2D Convolution LSTM being used in literature. From their research, they find that convolutional LSTM performs the best in capturing spatial temporal information, compared to LightGBM/XGBoost. Following that, all my efforts were focused on using and tuning ConvLSTM2D.

At one point during my model training, visualizing the model output in 2D made me see that my model tend to increase the demand values over time and also spread the demand outwards. I successfully resolved this by changing the last 3D convolution layer to have a 1x1x1 kernel from the original 3x3x3 kernel, and the performance becomes much better.

On a different note, a different approach of splitting the data into day-of-week and training separate LSTM models on each day-of-week was explored, but the results were not so promising, and by separating the data into day-of-week, we have much less data to train each LSTM model with, but might be a good idea to try out if more data is provided.



Visualising my model output helps me identify the problem. On the first column is the target output, and the second column is the prediction output.

We can clearly see the model is not performing well from this visualization, which is solved by modifying the last layer in the model.