FUNDAMENTALS OF DEEP LEARNING

Final Project

ARIVAZHAGAN Antosibirayan

MURUGESAN Bharanidharan

SUBRAMANI Praveen

5 April 2024

Predicting the Occupany Rate of cars parked in Birmingham.

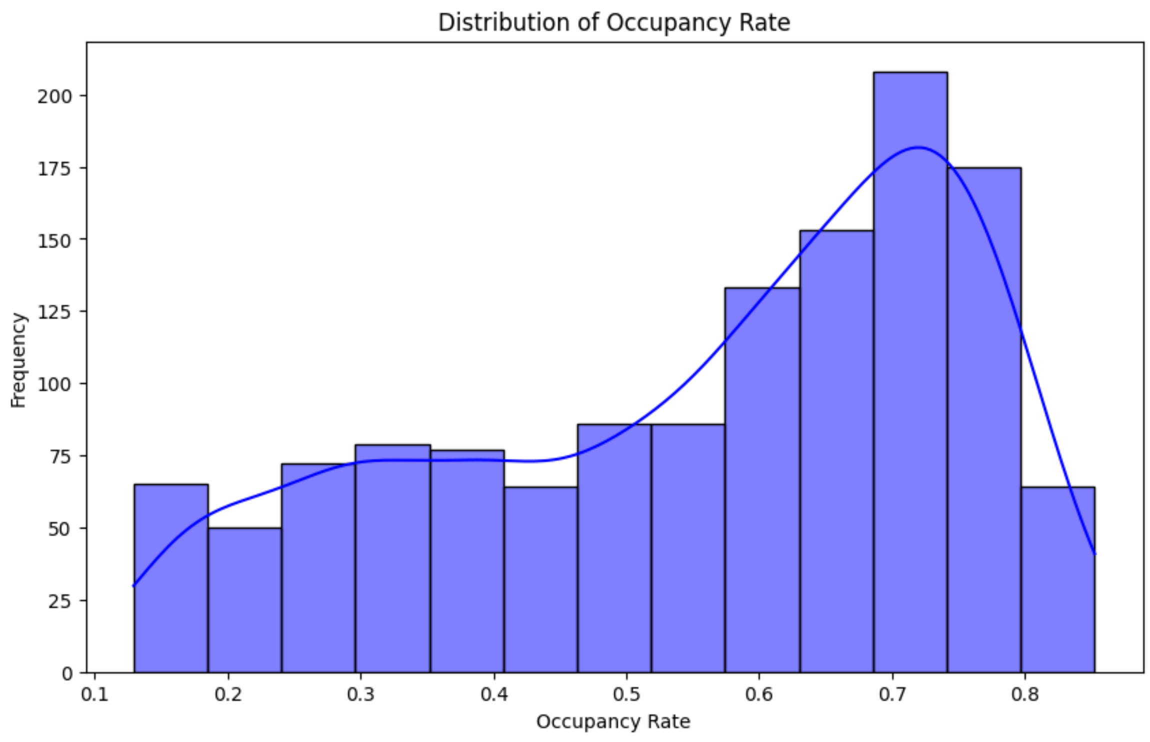
Description of Dataset:

The dataset comprises data collected from car parks in Birmingham operated by NCP, sourced from Birmingham City Council.

A focused subset of the dataset has been utilised. This subset was selected on the basis of the variable, SystemCodeNumber, where the values under this variable which had the string, "Shopping" w ere selected. Moreover, this subset comprises of data from October until December 19th, thus making it a sequential dataset.

Task Category of Dataset:

The dataset falls under the category of time series forecasting. In this exercise, the goal is to predict a single value (occupancy rate) based on historical data. Therefore, it would come under **many-to-one task category**.



This graph provides an overview of the occupancy rate data where the max value is 0.852604 and the min value stands at 0.129167.

Appropriateness of RNNs:

Recurrent Neural Networks are suitable for sequential data like time series because they can capture temporal dependencies effectively.
In this case, as the dataset involves time stamped data, that is, occupancy rate at different times, RNNs are appropriate for learning patterns and making predictions based on sequential information.

Data Preprocessing:

1. Created a new feature, Occupancy Rate by dividing the occupancy by capacity.
2. Dropped the columns, Occupancy and Capacity
3. Filtered the data to include only car parks with the system code containing 'Shopping'.
4. Split the data into train and test sets based on the 'LastUpdated' column where data for the months, October and November were taken as train set and the data from December 1st until 19th was taken as test set.

By meticulously evaluating the performance of each model architecture and its hyperparameters, we aim to identify the most effective model by comparing the RNN and CNN models. Our evaluation for the best model is based on the lowest RSME score for the respective models.

RNNs:

Recurrent Neural Networks are a class of neural networks particularly effective for handling sequential data. Unlike feedforward neural networks that can process each input independently, RNNs maintain a hidden state or rather internal state, that captures information about previous inputs in the sequence. This helps them to leverage temporal dependencies within the data.

Long Short-Term Memory: LSTM is a type of RNN architecture designed to address the vanishing gradient problem and capture long term dependencies much more effectively. It introduces a specialized memory cell that can maintain information over long sequences. LSTM have gates such as input, forget and output gates and these gates regulate the flow of information which allows them to selectively remember or even forget information from the past.

Gated Recurrent Unit: GRU is a simplified variant of LSTM with fewer parameters. It also incorporates gating mechanisms to control the flow of information but it merges the cell state and hidden state into a single state vector which leads to simplifying the architecture. GRU is computationally more efficient and easier to train compared to LSTM but at the same time, still being capable of capturing long-term dependencies.

Model Architecture Overview:

The models are based on multiple RNNs, based on LSTM and GRU architectures.

Different configurations of the RNNs are tested by varying numbers of layers and units per layer.

Hyperparameter Configurations:

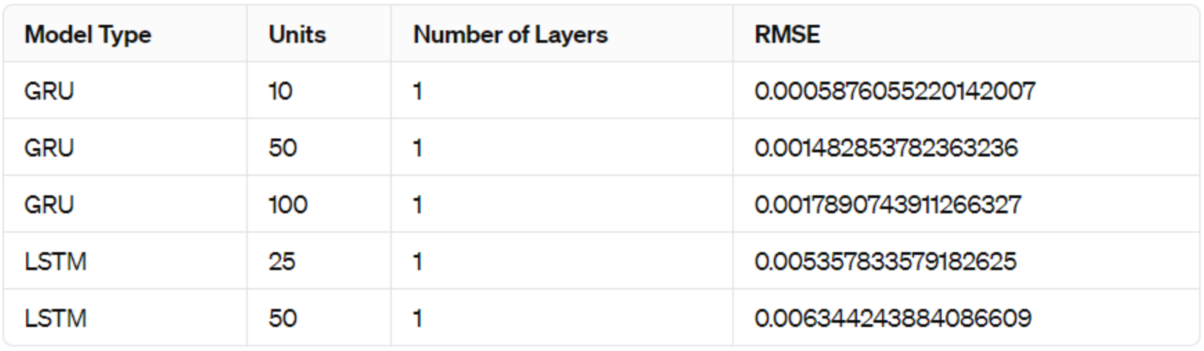
Each configuration is defined by three main hyperparameters:

model_type: Specifies whether the model is based on LSTM or GRU architecture.

units: Indicates the number of units (neurons) in each RNN layer.

num_layers: Specifies the number of RNN layers in the model.

Top 5 Models Based on Lowest RMSE:



Model Type	Units	Number of Layers	RMSE
GRU	10	1	0.0005876055220142007
GRU	50	1	0.001482853782363236
GRU	100	1	0.0017890743911266327
LSTM	25	1	0.005357833579182625
LSTM	50	1	0.006344243884086609

The top-performing model configuration, based on the lowest RMSE, is a GRU model with 1 layer and 10 units. This model achieved an RMSE of approximately 0.0006.

The GRU architecture with a single layer and 10 units outperformed other configurations, including LSTM models.

The simplicity of the GRU architecture with fewer parameters compared to LSTM contributed to its efficient performance in this dataset.

Training Models:

1. Input shape being fed into the LSTM or GRU layer within the Sequential model:
   The LSTM and GRU layers expect input data to have a 3-dimensional shape: (batch_size, timesteps, input_dim).
   Therefore, the following line of code was inputted,

   ```
   # Reshape input data
   X_train_reshaped = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
   X_test_reshaped = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))
   ```

   By adding np.expand_dims with axis=1, we are adding an extra dimension to represent the timestep for each data point. This ensures that the input data has the correct shape expected by the LSTM or GRU layer.

2. Early Stopping: has been implemented when training the models to automatically stop training when the validation loss stops improving, preventing overfitting and unnecessary training epochs.

Attention Mechanism:

In the create_rnn_model function, we have added an optional argument use_attention. When use_attention is set to True, it adds an attention layer to the last recurrent layer of the model. This allowed us to experiment with attention mechanisms alongside different RNN configurations.

By incorporating attention mechanisms into RNN architectures, the model gains the ability to dynamically focus on different parts of the input sequence based on the context, which can lead to improvements in performance.

Thus, adding an attention layer after the last recurrent layer allows the model to focus on specific parts of the input sequence which would improve its ability to capture relevant information for predicting the occupancy rate in the given dataset.

Moreover, we have included a few graphs in the code to describe attention weights.

CNN for 1D Input:

Convolutional Neural Networks for 1D input are commonly used for sequence data processing, particularly in tasks like time series analysis. In 1D CNNs, convolutional layers slide across the input sequence and extracts local patterns. Pooling layers reduce the dimensionality of the features extracted by the convolutional layers. Dense layers are typically added after the convolutional layers to perform regression or even classification tasks.

Comparing RNN and CNN Performance:

The Root Mean Squared Error of the best performing RNN model is 0.0013, while the RMSE of the CNN model is 0.0003.

The CNN model outperforms the RNN model in terms of RMSE, which indicates it provides better predictions for the occupancy rate of car parks in Birmingham.

Differences in Architecture between RNN and CNN:

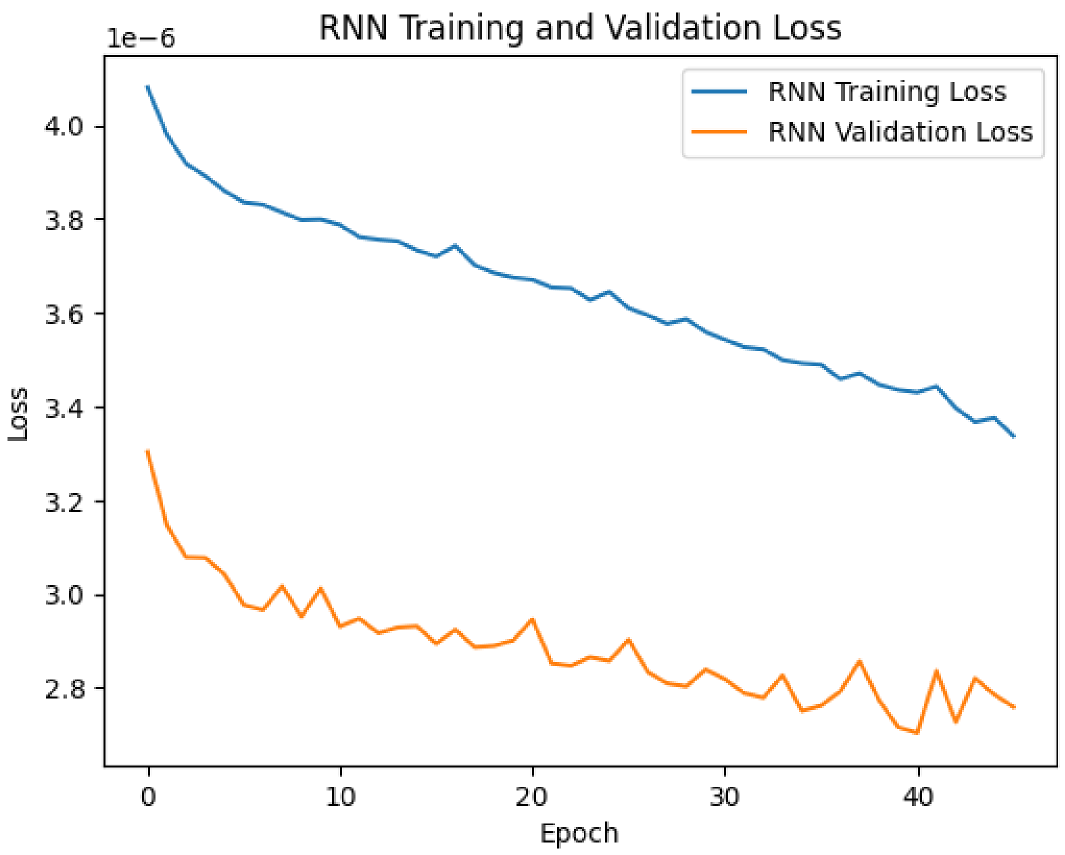
For RNNs,

- RNN architectures, such as LSTM and GRU, are designed with recurrent connections that allow them to maintain an internal state and handle sequences of varying lengths.
- This architecture enables RNNs to remember past information while processing current inputs making them good for tasks where historical context is crucial for making accurate predictions.

For CNNs,

- CNN architectures typically consist of convolutional layers that slide across the input space, extracting spatial patterns.
- While CNNs are primarily designed for grid like data such as images, they can also be adapted for sequential data processing such as in this case, occupancy rate prediction.

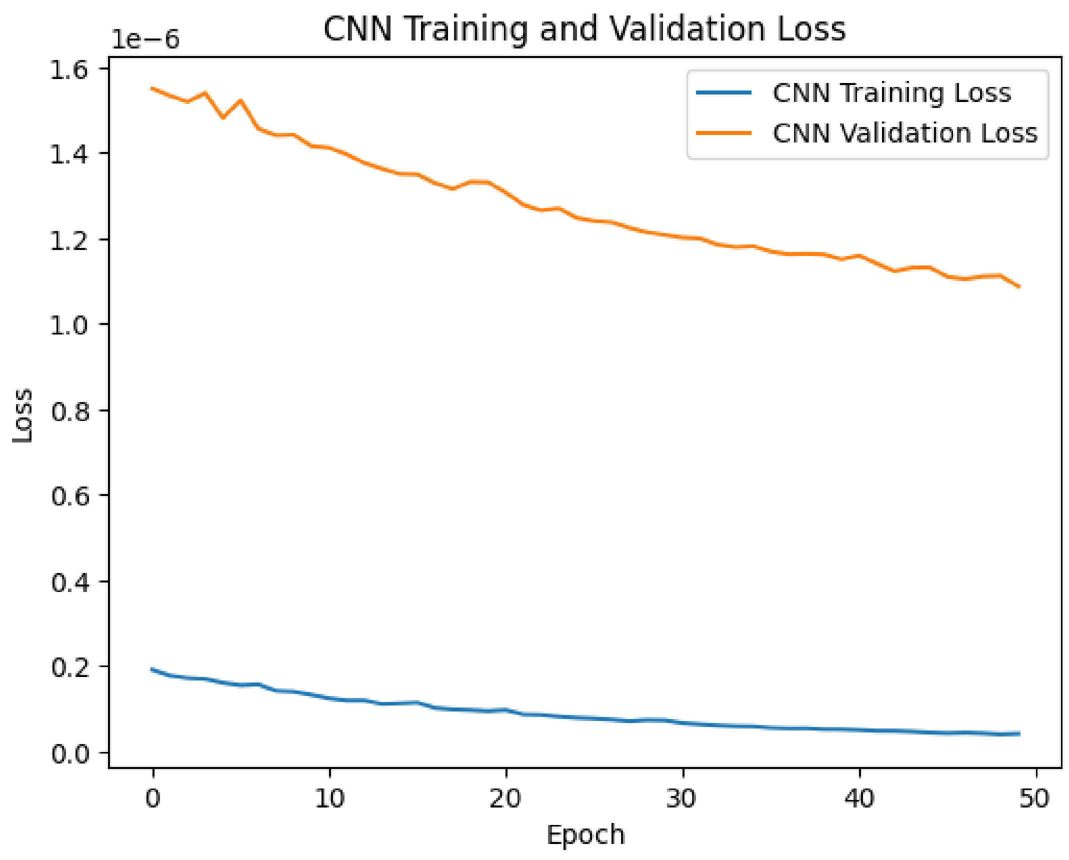
Training and Validation loss for RNN:



If the training loss is still decreasing after a certain number of epochs, it indicates that the model is still learning and may benefit from additional epochs. Conversely, if the training loss has starts to increase, it suggests that the model may have converged, and further training epochs could lead to overfitting.

In our case, there are small spikes and then downs which means there is a low amount of overfitting as the number of epochs.

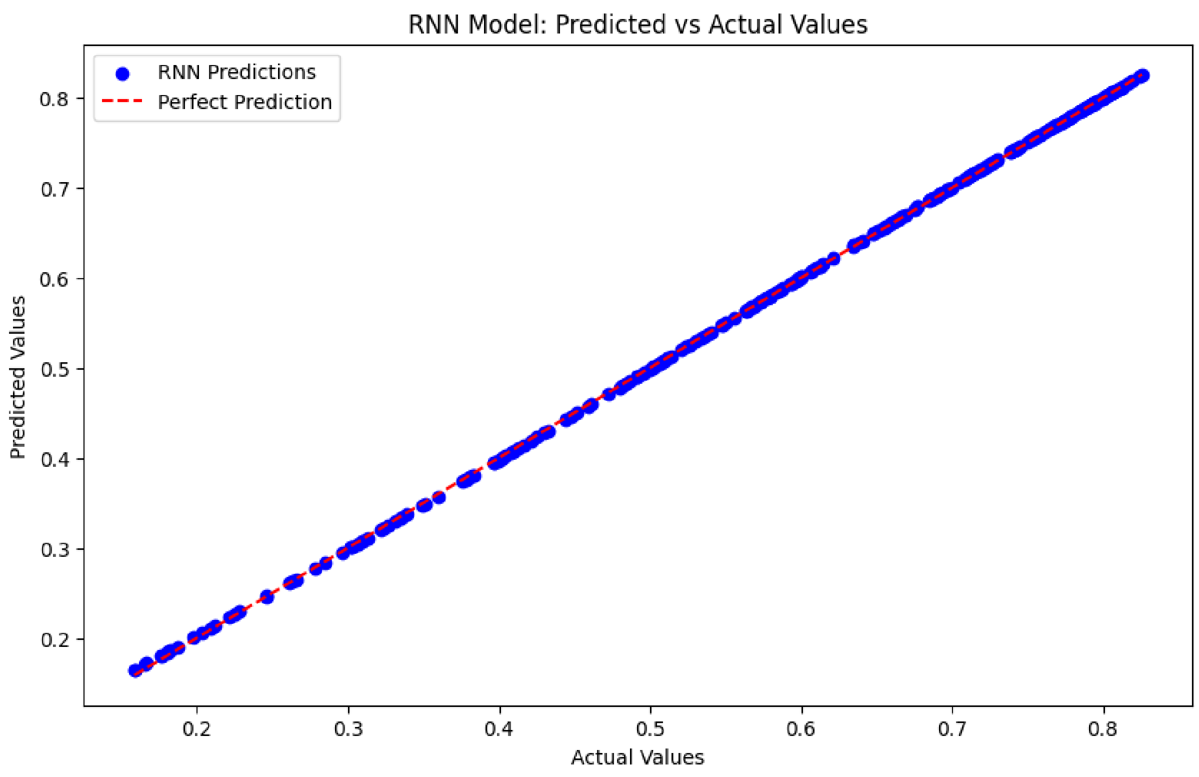
Training and Validation loss for CNN:

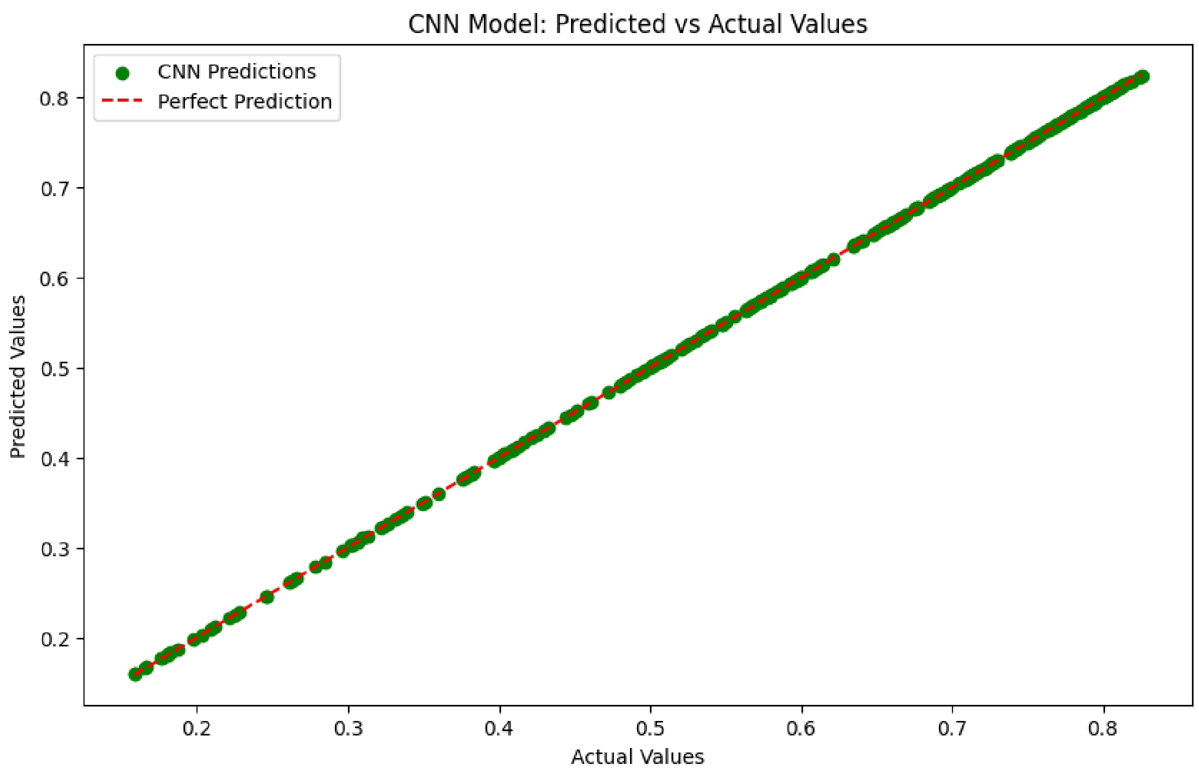


If the validation loss starts to increase after a certain number of epochs, it indicates overfitting, and training should be stopped or the number of epochs should be reduced. On the other hand, if the validation loss continues to decrease or remains stable, it means the model is good.

In our case, the graph represents no overfitting.

Predicted vs Actual Values:





Sources :

https://archive.ics.uci.edu/dataset/482/parking+birmingham

https://www.bluecourses.com/courses/course-v1:bluecourses+BC10+2020_Q4/course/

https://keras.io/api/applications/

https://towardsdatascience.com/cnn-architectures-a-deep-dive-a99441d18049

https://mycourses.ieseg.fr/fundamentals_of_deep_learning

2004.14545.pdf (arxiv.org)

https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/keras/layers/recurrent.py#L2317

https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/keras/layers/recurrent.py#L2744

ChatGPT