MASTER OF SCIENCE DISSERTATION

Dissertation Title:	Data Highways: Development and Analysis of a Deep Neural Network for Traffic Prediction in Oxford, UK
Surname:	Di Carlo
First Name:	Luca
Student Number:	19137568
Supervisor:	Dr. Shumao Ou
Dissertation Module:	DALT7017
Course Title:	Dissertation in Data Analytics
Date Submitted:	September 24, 2021

Statement of Originality

Except for those parts in which it is explicitly stated to the contrary, this project is my own work. It has not been submitted for any degree at this or any other academic or professional institution.

Signature of Author		Date	**14/09/2021**

Regulations Governing the Deposit and Use of Master of Science Dissertations in the School of Engineering, Computing and Mathematics, Oxford Brookes University.

1. A copy of dissertation final reports submitted in fulfilment of Master of Science course requirements shall normally be kept by the School.

2. The author shall sign a declaration agreeing that, at the supervisor's discretion, the dissertation will be submitted in electronic form to any plagiarism checking service or tool.

3. The author shall sign a declaration agreeing that the dissertation be available for reading and copying in any form at the discretion of either the dissertation supervisor or in their absence the Programme Lead of Postgraduate Programmes, in accordance with 5 below.

4. The project supervisor shall safeguard the interests of the author by requiring persons who consult the dissertation to sign a declaration acknowledging the author's copyright.

5. Permission for anyone other than the author to reproduce in any form or photocopy any part of the dissertation must be obtained from the project supervisor, or in their absence the Programme Lead of Postgraduate Programmes, who will give his/her permission for such reproduction only to the extent to which he/she considers to be fair and reasonable.

I agree that this dissertation may be submitted in electronic form to any plagiarism checking service or tool at the discretion of my project supervisor in accordance with regulation 2 above.

I agree that this dissertation may be available for reading and photocopying at the discretion of my project supervisor or the Programme Lead of Postgraduate Programmes in accordance with regulation 5 above.

Signature of Author		Date	**14/09/2021**

Data Highways: Development and Analysis of a Deep Neural Network for Traffic Prediction in Oxford, UK

Luca Di Carlo (19137568)

Program: MSc Data Analytics
Module: DALT7017 Dissertation in Data Analytics
Submission: Dissertation Report
Academic Year: 2020-2021
Word Count: 9826
Number of Illustrations: 28

Links to project source code:

GitHub:
https://github.com/ldicarlo1/development_of_traffic_model_for_Oxford

Google Drive:
https://drive.google.com/drive/folders/1oOU4rlTzUmQ8JtTFbQ0AZkdqnwIZ6Bzg?usp=sharing

Abstract

In today's modern era, the convenience and prevalence of transportation is directly proportional to the pace at which urban cities are growing across the world. However, the unprecedented growth of these cities poses a massive logistic, economic, and environmental problem particularly within the issue of traffic congestion. A mitigation for this has been resolved by smart cities which utilize traffic prediction to effectively manage and prevent traffic congestion. Research in the field of traffic prediction is constantly evolving, with a large focus on machine learning models and deep learning to provide accurate traffic predictions. Deep learning has proved very successful in the field of traffic prediction, typically involving deep networks with large sources of data which power these structures. Despite many deep learning methods available, the sources of historical traffic data remain very scarce throughout the field of research. In this dissertation, a method for collecting commercial traffic and weather data was developed in addition to a deep hybrid neural network approach to modeling this data for a local area of Oxford, UK. The proposed deep network architecture was compared against benchmark methods, and the integration of weather data was compared as well. Varying prediction horizons were also compared to determine a suitable proof-of-concept predictive traffic model for Oxford. Overall model performances found the proposed traffic model to be a good predictor of short-term traffic speeds, and several improvements are discussed to continue the work of this project.

Contents

Acknowledgements

I would like to thank Dr. Shumao Ou for his supervision of this dissertation and guidance which was very helpful to the completion of this project.

I also would like to thank the lecturers at Oxford Brookes University who provided me with the base knowledge and have been valuable to this dissertation and my professional life.

Finally, I would like to thank my family, friends, and fellow students who have provided me with support throughout this difficult year considering the Covid-19 pandemic and the difficulties faced by students alike.

1. Introduction

Traffic prediction is a problem that every urban city is attempting to solve in today's era of big data. Every major commercial transportation provider has a form of traffic prediction, including TomTom (Laranjeira, 2020), Google Maps (Lau, 2020), and HERE Traffic (DeLancey, 2018).

1.1 Purpose and Motivation

1.1.1 Smart City Impact

Smart cities today rely heavily on traffic prediction to improve transportation times and reduce traffic congestion. There are two major reasons for this: faster travel allows cities to be more efficient and is beneficial to residents, and less vehicle congestion results in less overall carbon emissions and contributes to the sustainability of modern urban areas. As cities grow, these factors will only worsen, hence creating an opportunity for innovative modelers to capture these patterns and provide real-time and future solutions. Intelligent Transportation Systems (ITS) seamlessly integrate data from different modes of transportation to provide end users with better traffic management strategies and timely transportation planning within urban areas. Improving these systems with short-term traffic prediction would significantly improve the transportation experience for government authorities, the private sector, and most importantly the individuals on the roadways.

1.1.2 Autonomous Vehicles Impact

Whilst cities continue to grow, so will the number of autonomous vehicles on the road. Self-driving vehicles will only improve over time, and eventually traffic patterns and forecasts will need to be integrated into the decision making of these vehicles' software to make travel more efficient. A study by Suh, Shao and Sun (2020) found that connected autonomous vehicles improved short-term traffic prediction by 25% by collecting real-time data on roadways, which can then in turn allow for the autonomous

vehicles to make intelligent routing decisions. Improved traffic prediction would allow the networks of these vehicles to work collectively in reducing traffic congestion. A survey by Miglani and Kumar (2019) explored the current field of traffic prediction for autonomous vehicles, highlighting that short-term traffic prediction is key to reducing travel times amongst automated cars, with a particular focus on deep learning models.

1.2 Aim and Objectives

The aim of this research is to develop a proof-of-concept traffic model base architecture to make short term traffic speed predictions for the Peartree Roundabout in Oxford.

The objectives can be described as the following:

- Collect and store commercial traffic and weather data for Oxford.
- Develop a deep hybrid neural network base architecture for short-term traffic prediction which incorporates both traffic and weather data.
- Implement hyperparameter tuning to train and test the candidate model.
- Critically assess and analyze the candidate model performance against other benchmark models.
- Determine and discuss methods or developments that would improve the candidate model for future research.

1.3 Existing Traffic Prediction Models for Oxford

There currently is a traffic model in development for the locality of Oxfordshire named MIMAS, which provides predictions into the number of cycles, buses, cars, and trucks for any given day (David, 2020). However, there is no mention whether this model predicts in the short-term or long-term or whether it contains weather data. This traffic model is in current use by the Oxfordshire County Council Public Transportation department.

1.4 Structure

The structure of this report will have the following sections: a discussion on the background and theory of traffic prediction, a literature review of the current methods of traffic prediction, a brief overview of the tools that were used in this research, a description of the data collected and the process of collection, a detailed overview of the methodology, results and analysis, and a conclusion and discussion.

1.5 Graphs and Visualizations

All graphs and visualizations in this paper were produced using the following Python libraries: matplotlib (Hunter, 2007), seaborn (Michael and Olga, 2017), and Tableau ('Tableau (version. 9.1),' 2016).

2. Background and Theory

2.1 Traffic Prediction

Traffic prediction is defined as the use of historical traffic patterns to predict future patterns *N* number of time steps into the future. There are two main types of traffic prediction: traffic flow and traffic speed. Traffic flow is the total number of vehicles that transverse a given road over a fixed period. Traffic speed is the average traveling velocity of all the vehicles in each road space. The focus of this research is predicting traffic speeds for a locality of Oxford, specifically the Peartree Roundabout. Most research currently exists in the field of short-term traffic prediction, which is forecasting traffic speeds 5 to 60 minutes into the future and this will be adapted into this study (Tedjopurnomo *et al.*, 2020).

The traffic prediction problem can be described as below:

$$\widehat{y}_{t_{+T'}} = f([X_{t-T+1}, X_{t-T}, \dots, X_t])$$

- $\boldsymbol{y_t}$: The observed traffic time at time t
- $\boldsymbol{\widehat{y}_t}$: The predicted traffic time at time t
- $\boldsymbol{T}$: The input sequence length (the total number of time steps in the past that is used as the input)
- $\boldsymbol{T'}$: The prediction horizon (how many steps into the future the prediction is for)

Equation 1: Traffic prediction problem equation (Tedjopurnomo et al., 2020).

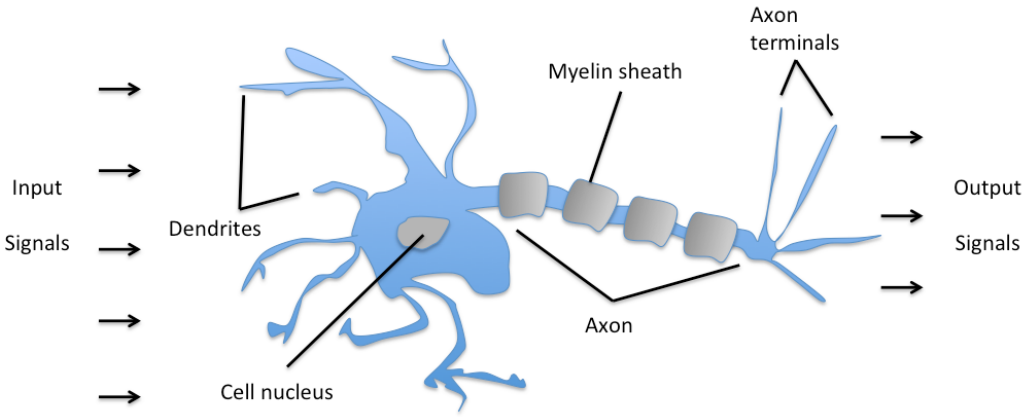
Traffic prediction is a spatiotemporal problem which revolves around a network of roads, representing the spatial domain, and forecasting the speeds on each road of the network, which is the temporal domain. Most of the emphasis lies in the temporal domain which dominates the traffic speeds due to commute times, weather, and accident impacts. However, the spatial domain is important for short-term traffic prediction, which relies on understanding the traffic road-network topology to predict areas of high traffic and how they interact with the surrounding roads (Yin, 2021). For example, during times of high traffic vehicle operators may take alternative routes that would result in increased traffic volume in secondary roads. Understanding the road network topology is key to developing a flexible, predictive traffic model that can adapt to real-time variables such as weather or accidents.

2.2 Neural Networks

2.2.1 Single Layer Perceptron

A single layer perceptron is the building block of a neural network containing a neuron, a weight, and an activation function and was inspired after the neurobiological structure of a neuron within a brain. A biological neuron can be explained using Figure 1, which

diagrams the flow of information through a neuron: input signals are absorbed by dendrites, which pass the signal to the cell nucleus where the cell state is determined as active or inactive, and if active the signal is output through the axon to a neighboring cell to repeat the process (Raschka, 2015).



Schematic of a biological neuron.

Figure 1: A biological neuron diagram (Raschka, 2015).

Similarly, this process is represented in the artificial neuron: input data is fed into the neuron where it is weighted and summed, which then is fed through an activation function. The activation function receives a linear combination of the inputs and will output either 1 or -1, which will determine the state of neuron as “active” or “dormant” (Ujjwal, 2016). This process is well represented in Figure 2 where x_i is the input sequence and w_i is the weight.

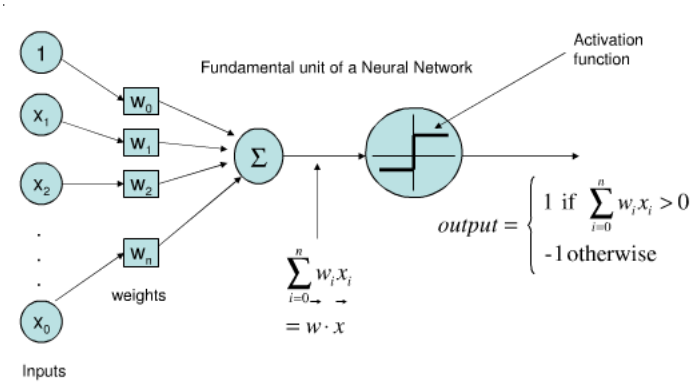


Figure 2: Neural network diagram (Verma and Singh, 2015).

Using Rosenblatt's initial perceptron rule the weights can be updated for each training sample x_j^i accordingly using the output value, the target value, and learning rate n which is shown in Equation 2 (Rosenblatt, 1957).

$$\Delta w_j = n\left(target^{(i)} - output^{(i)}\right) x_j^i$$

- $\boldsymbol{x_j^i}$: Each training sample
- $\boldsymbol{n}$: Learning rate (between 0 and 1)
- $\Delta \boldsymbol{w_j}$: Change in weight

Equation 2: Calculation of the updated change in weight (Raschka, 2015).

As a network learns over each epoch, a loss function and gradient descent work together to minimize loss. The rate at which the network learns is called the learning rate. This parameter is between 0 and 1 and sets the magnitude at which the weights are updated to locate the point of optimal loss called the global minima. Using a value too large can result in divergent behavior in the loss function and cause the model to improperly learn, additionally the network may not achieve global minima. A value too small will result in a network taking too long to achieve the global minima. An optimal learning rate will achieve global minima in a reasonable time without compromising the learning of the network (Figure 3). Learning rates are commonly initialized at the value 0.001 as a base before changes are made (Park, Yi and Ji, 2020).

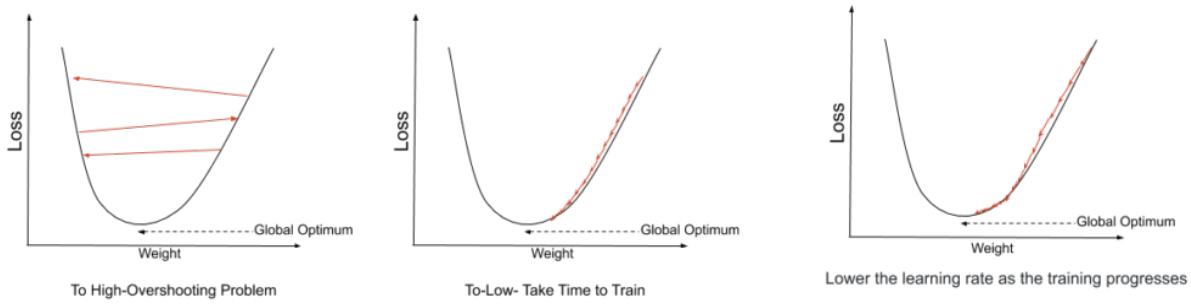


Figure 3: Different learning rates and their performances (Jordan, 2018).

2.2.2 Multi-layer Perceptron

A multi-layer perceptron contains multiple layers of single layer perceptrons, and each layer of perceptrons may contain many neurons. In Figure 4 there is an input layer of neurons, which directly feed into a hidden layer of neurons to eventually produce a prediction y. This feed-forward stage can be referred to as forward propagation, whilst the stage where the weights are updated based on the error between the target and predicted class is called the back propagation phase (Gardner and Dorling, 1998).

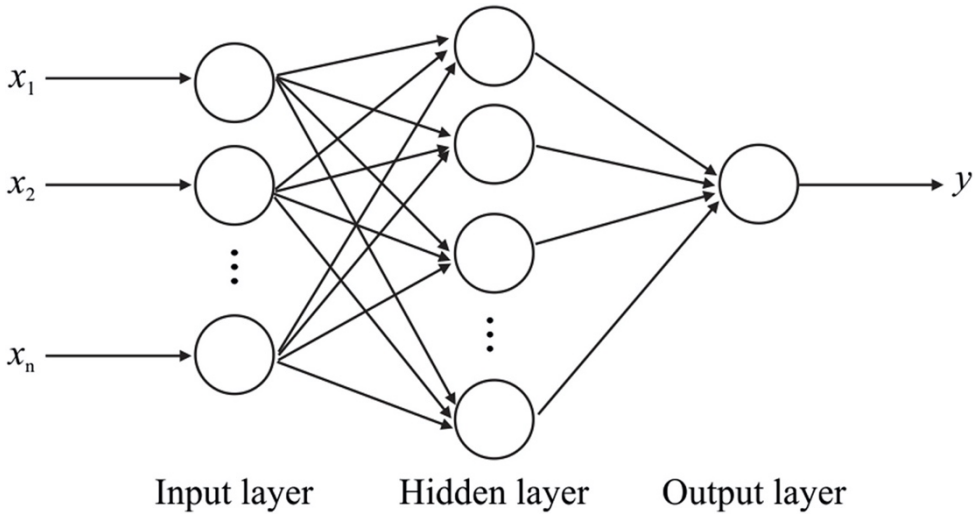


Figure 4: A deep neural network architecture (Hashemi Fath, Madanifar and Abbasi, 2020).

As this process repeats, the network “learns” and simultaneously attempts to minimize prediction error by updating the weights accordingly. The function which evaluates the

error produced during each process of forward and back propagating is called the loss function. The optimization algorithm that minimizes the loss function is called gradient descent, which attempts to achieve a minimal value of error (David and James, 1987).

In supervised learning, a neural network is trained using features and a target variable with each forward and back propagation through the entire training dataset being called an epoch. Over each epoch a loss value is produced using gradient descent which in a correctly organized network should decrease as the model learns. Typically, a neural network will calculate training loss (error on the training data) and a validation loss (error on the validation data) (Günther and Fritsch, 2010). As training occurs, training loss should decrease along with validation loss over each epoch. Should validation loss begin to increase this would indicate model overfitting, suggesting that the model would not perform as well on unseen data. This is well represented in Figure 5, where the "sweet spot" is defined as the number of epochs where validation and training loss is globally minimized (Salehinejad *et al.*, 2017).

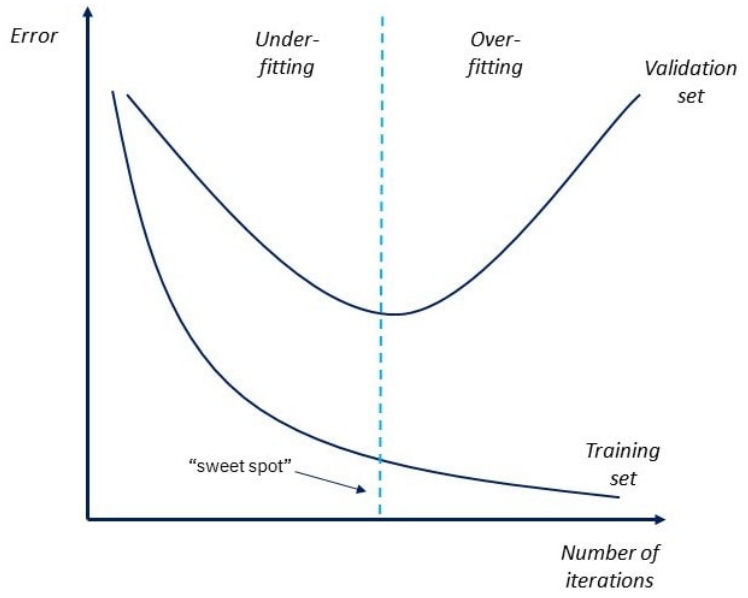


Figure 5: Diagram of underfitting and overfitting a neural network (IBM and Education, 2021).

2.2.3 Batch Normalization

Batch normalization is a standardization technique which is applied between layers of a neural network. As batches of information are propagated between layers of a neural network, the outputs of a layer are standardized to a standard deviation of one. This stabilizes the training of a neural network and also may speed up the process (Ioffe and Szegedy, 2015).

2.2.4 Dropout

To prevent networks from overfitting too soon, adding a dropout layer within a neural network can mitigate this issue. A dropout layer simply removes units from a layer which "thins" the relationships between layers and prevents the network from learning the connections too quick (Srivastava *et al.*, 2014).

2.3 Recurrent Neural Networks

2.3.1 Basic Structure

A recurrent neural network (RNN) is a type of neural network that has an internal "memory" that can remember previous state outputs as inputs to whilst also having hidden states. This allows RNNs to effectively model time series data and natural language processing since the model will retain a memory of previous inputs or words when making a new prediction. A standard RNN can be thought of as a chain of multi-layer perceptrons, that when fed time series data makes a prediction on each time step and uses each prediction over time to make new prediction (Salehinejad *et al.*, 2017). When back propagation occurs, error is calculated across time, and weights are also updated over each time step. Figure 6 is a diagram of an RNN that is unrolled, with h_t representing each output over time, x_t representing the input sequence, and A representing the network (Niklas, 2021).

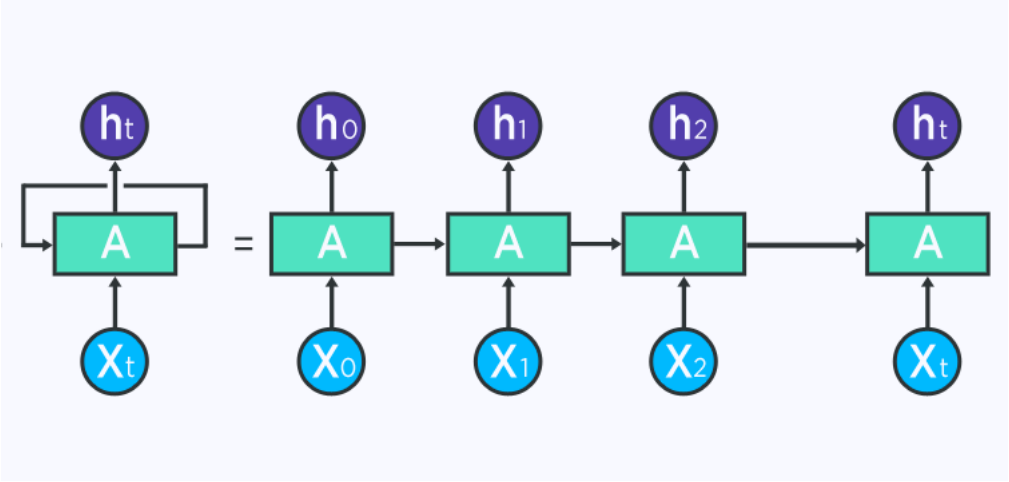
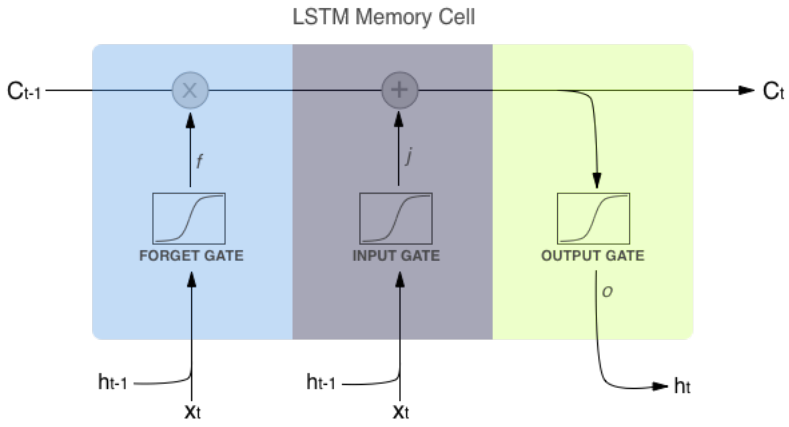


Figure 6 : An "unrolled" RNN, which is a chained group of neural networks (Niklas, 2021).

2.3.2 LSTM

A long short-term memory network (LSTM) is an extension of an RNN, which improves the long-term memory of said networks. An LSTM uses three gates: an input gate, an output gate, and a forget gate to make predictions. Each gate uses a sigmoid function to produce a value between 0 and 1, with "0" closing the gate and "1" opening the gate (Hochreiter and Schmidhuber, 1997).



- C_{t-1} : Cell state from previous LSTM cell t-1.
- h_{t-1} : Output from previous LSTM block at timestep t-1.
- x_t : Input at current timestamp.
- C_t : Cell state from current timestamp.

Figure 7: An LSTM cell diagram (Thakur, 2018).

Each LSTM cell state is calculated by determining what information needs to be forgot from the previous cell state and what needs to be considered from the current cell state. Repeating this process over many time steps allows the LSTM network to remember key long-term patterns whilst also still considering the short-term inputs (Staudemeyer and Morris, 2019).

The three equations for each gate can be found in Equation 3.

$$i_t = \sigma(w_i[h_{t-1}, x_t] + b_i)$$
$$f_t = \sigma\big(w_f[h_{t-1}, x_t] + b_f\big)$$
$$o_t = \sigma(w_o[h_{t-1}, x_t] + b_o)$$

- $\boldsymbol{i_t}$: Input gate.
- $\boldsymbol{f_t}$: Forget gate.
- $\boldsymbol{o_t}$: Output gate.
- $\boldsymbol{\sigma}$: Sigmoid function.
- $\boldsymbol{w_x}$: Weight for the respective gate neurons.
- $\boldsymbol{b_x}$: Bias for each gate.

Equation 3: Three equations for the input, forget, and output gates of an LSTM (Thakur, 2018).

2.4 Graph Neural Networks

2.4.1 Graph

A graph is data structure that can be represented as a set of nodes and edges, in which the nodes are connected by edges. Each node contains information, and edges may

contain weights. An adjacency matrix defines the connections between nodes in a graph structure, and the feature matrix contains all the information stored in the nodes. The simple expressions to represent a graph can be found in Equation 4.

$$G = (V, E)$$
$$A \in R^{n\,x\,n}$$
$$X \in R^{n\,x\,d}$$

- $\boldsymbol{G}$: The graph.
- $\boldsymbol{V}$: The vertices (nodes).
- $\boldsymbol{E}$: The edges.
- $\boldsymbol{A}$: The adjacency matrix.
- $\boldsymbol{X}$: The feature matrix of a graph.
- $\boldsymbol{R}$: The feature vector.
- $\boldsymbol{d}$: The dimension of a node feature vector.
- $\boldsymbol{n}$: The number of nodes.

Equation 4: Graph structure equation (Wu et al., 2021).

Graphs may be directed or undirected, dependent on the functionality of the data structure. In a directed graph, all edges are directed from one node to another in an organized directional manner. Undirected graphs connect all edges and the flow of information through the graph can flow both forwards and backwards, with the direction ignored (Wu *et al.*, 2021).

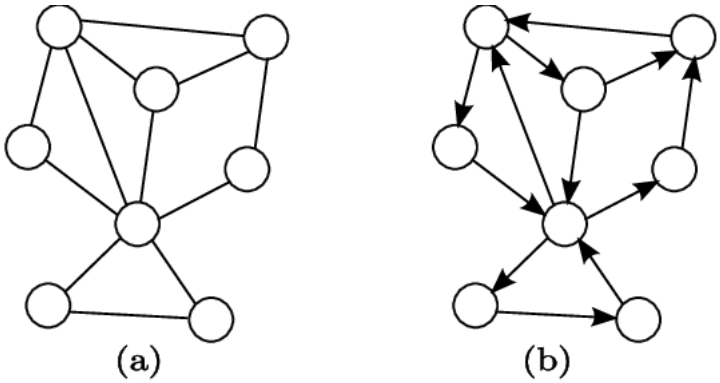


Figure 8: Examples of an undirected (a) and directed (b) graph (Fionda and Palopoli, 2011).

Graphs also may be spatiotemporal, which is when the node attributes change dynamically over time (Jain *et al.*, 2015).

$$G^t = \left(V, E, X^{(t)}\right)$$
$$X^{(t)} \in R^{n\,x\,d}$$

- $\boldsymbol{G}$: The spatiotemporal graph.
- $\boldsymbol{X}$: The feature matrix of a spatiotemporal graph.

Equation 5: A representation of a spatiotemporal graph (Jain et al., 2015).

2.4.2 Graph Convolutional Neural Networks

A graph convolutional neural network (GCN) is a neural network that can perform convolutional operations on organized graph structures. GCNs are extremely effective on image or spatial datasets, and currently have spectral and spatial-based approaches. The main focus for this research is the spatial-based approach (Wu *et al.*, 2021).

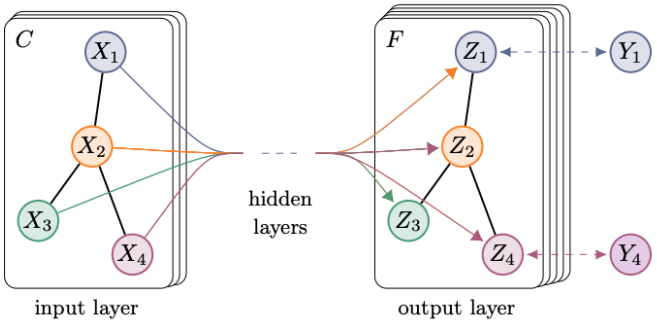


Figure 9: A visual representation of a GCN (Kipf and Welling, 2017).

The development of the mathematical representation of a GCN is based in graph signal processing. For GCNs, an undirected graph is assumed which is a Laplacian matrix, which in the Fourier domain can be eventually factored into a multi-layer GCN with the following formula in Equation 6.

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)})$$

- $\tilde{\boldsymbol{A}}$: The adjacency matrix of an undirected graph.
- $\boldsymbol{W}^{(l)}$: A layer-specific trainable weight matrix.
- $\boldsymbol{\sigma}$: An activation function.
- $\boldsymbol{H}^{(l)}$: A matrix of activations for layer l
- $\tilde{\boldsymbol{D}}$: A diagonal matrix of node degrees, where $D_{ii} = \sum_j(A_{i,j})$

Equation 6: A GCN expression (Kipf and Welling, 2017).

3. Literature Review of Traffic Prediction

3.1 Historical Methods

There are three major approaches to traffic modeling that have been developed over the past fifty years: classical statistical methods, machine learning, and deep learning (Tedjopurnomo *et al.*, 2020). Classical statistical methods were initially used by

modelers, however they assume stationarity of the traffic data and fail when presented with the complex, non-linear nature of real traffic data (Yuankai and Tan, 2016). Williams and Hoel (2003) used a seasonal ARIMA model which performed well on temporal data alone, with no spatial road network considered. The deficiencies of classical statistical methods led researchers to machine learning methods, namely neural networks. Machine learning methods are more flexible than statistical methods, given that they adjust and update their parameters to better fit non-linear data as they are trained (Warner and Misra, 1970). In the modern day, the combination of more data and the easier implementation of neural networks in programming languages has resulted in the widespread use of deep hybrid neural networks for traffic prediction (Tedjopurnomo *et al.*, 2020).

3.2 Modern Methods

The current methods today capture the spatial and temporal aspects of traffic forecasting, as well as the non-linearity of traffic data by using variations of feedforward neural networks (Tedjopurnomo *et al.*, 2020). Many of the methods today use convolutional neural networks (CNN) to capture the spatial component of the road network, and a form of RNN for the temporal component. Sun, Wu, and Xiang (2020) used a two-dimensional CNN on city-wide road network to predict traffic flow. Yuankai and Tan (2016) used convolutional neural network CNN in combination with two LSTM layers to model a segment of highway across southern California. The CNN captured the spatial aspect of the data, convoluting across the road network whilst the LSTM cell mitigated the temporal variation in the traffic data which yielded promising results. Conversion of the road network to a Euclidean grid allows for the a CNN to capture the road topology easier, however road networks are seldom uniformly spaced and have non-Euclidean dependencies (Lee *et al.*, 2021). Mapping a road network to a matrix requires time-consuming algorithms to transform the data, additionally many times data is lost during the transformation (Jia and Yan, 2020).

Graphs are excellent for capturing non-Euclidean data structures and can describe the topological structure of road networks better than CNNs. A road segment graph is graph

that assigns each road trajectory to a node, and this method of data organization has been adopted by much of the latest research in traffic modeling (Wu *et al.*, 2021). In addition, graph-based convolutional architectures have been found to greatly reduce the number of parameters in a neural network without worsening the test error (Bruna *et al.*, 2013). Yuan *et al.* (2021) converted a portion of the road network of Houston to a graph for forecasting road flooding, with very good performance. Guo *et al.* (2019) developed a novel spatial-temporal graph convolutional neural network (GCN) which performed well on cyclical traffic flow data. A temporal GCN (T-GCN) architecture was developed by Zhao *et al.* (2019) which combined a GCN and a gated-recurrent unit (GRU), which is similar to an LSTM cell with only two gates. However, the GRU implementation was chosen due to its simpler implementation and quicker performance, with nearly equal performance. Using 1-hour of 15-minute intervals as an input sequence, the prediction lengths of 15, 30, 45, and 60 minutes were compared amongst models with the T-GCN overall being the superior model.

3.3 Incorporating Secondary Data

One of the major advancements of hybrid neural networks is their ability to incorporate data from multiple sources to make predictions. In the case of traffic prediction, there is a small amount of available literature that incorporates external data such as weather, twitter, or live video footage. One of the major reasons for this is secondary data sources tend to have little or no impact on overall traffic performance. In addition, many varying architectures have been tested but there is no consensus on which is best. For example, Soua, Koesdwiady, and Karray (2016) collected traffic, weather, and tweet data to incorporate into a deep belief network using data fusion as final step of their modeling procedure. Conversely, Essien *et al.* (2021) incorporated the same data types and used data fusion as the first step of his model, achieving similar results. Both concluded weather data is a weak predictor, however it did attribute to slightly better model performance. Zhang and Kabuka (2018) took a different approach, combining the data into a GRU cell a in high dimensional matrix. This approach however fails to adequately model the spatial domain and the relationships between the roads in the

road network and assumes the weather variables are universal for all the roads in the network.

4. Tools

4.1 Programming Languages

Across the number of traffic prediction papers with opensource code, Python is the only language of use (Lee *et al.*, 2021). The reason for this can be attributed to the number of machine learning libraries that can be easily implemented in very few lines of code.

4.2 Python Libraries

The three most common libraries are Pytorch, keras, and Tensorflow (Chollet and others, 2015; Abadi *et al.*, 2016; Paszke and Adam and Gross, 2019). Keras provides users with simple syntax to build neural networks and scale them up very quickly. Tensorflow and keras are both compatible, as many of keras's dependencies require Tensorflow. Pytorch is a scientific library that allows users to build their networks from scratch, making it machine learning more customizable.

In this dissertation, Python was chosen as the main programming language alongside many of its libraries.

5. Data

For this research, traffic and weather data needed to be collected in real-time for a locality of Oxford over a set period. The process required two separate sources of data and code which are discussed in this section, along with the rationale and specifics. Additionally, brief analyses of the two datasets are discussed and visualized.

5.1 Data Description

5.1.1 Traffic Data Description

The entirety of traffic prediction papers share publicly available datasets that collect data from in-situ road network environments such as traffic sensors. Oxford city does not have any public datasets, and this prompted the decision to collect data from a commercial traffic provider. HERE traffic API provides users with live traffic speed data on a minute-by-minute basis across Europe and North America. There is little documentation on how this data is collected, besides it being dynamically collected from a variety of sources including sensors, video footage, electronic devices, and traffic reports (Navmart, 2021). This data is only available through a live source, therefore this data was to be collected over a period of continuous data collection. The HERE traffic API is publicly accessible and provides users with 250,000 free data transactions before charges are incurred. Traffic data is provided as real-time traffic speeds for a stretch of road, which on main roads is typically two lanes (opposite directions of travel) but on secondary roads this is reduced to one lane of averaged speed for both directions. Traffic speeds were collected in km/h and missing values are reported as -1.

5.1.2 Location of Traffic Data Collection

There are 2550 road segments in Oxford city, each of which containing unique speed data. Incorporation of all the road data in Oxford would have been ideal, however the amount preprocessing time required to organize the spatial structure of the road network would have been too much for the scope of this research. One of the drawbacks to using spatial data is converting the data to a spatial representation (a graph or spatial matrix) since very few algorithms exist which can effectively convert the data, with most of the preprocessing being done by hand. Because of this constraint, only a small subset of the Oxford city road network was chosen. The candidate location needed to satisfy three criteria: high volume of traffic, complex spatial road design, and have dynamic changes in traffic throughout the week. The Peartree Roundabout, located on the intersection of the A34 and the A44 in northwest Oxford satisfied these conditions. Many commuters from the surrounding towns use this roundabout to enter Oxford or to enter on the A34 to travel to other part of the city. Figure 10 is a graphical representation of the road network, which includes a secondary sister roundabout that

connects Woodstock Road to the Peartree roundabout. In this candidate road network, there are 70 road segments which is a significant reduction from the original 2550.

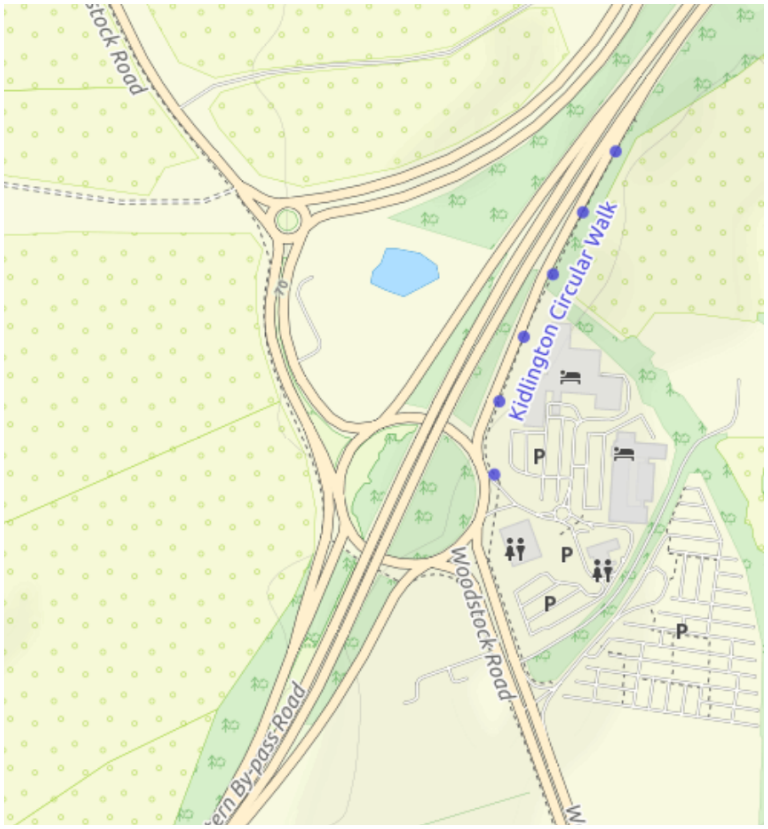


Figure 10: The Peartree Roundabout in Oxford (Source : viamichelin.co.uk).

5.1.3 Amount of Traffic Data Collected

Most research into traffic prediction use typically 6-12 months of data. This is to provide machine learning models with ample data to train on, to capture the seasonal change in patterns, and to gauge model performance in different times of the year (Tedjopurnomo *et al.*, 2020). However, due to the scope of this research a shorter period of data collection was employed, mainly because of time constraints and this being a proof-of-concept. A virtual machine was chosen to collect data continuously over a three-week period since it could be left running without disturbance during this time. Data collection was delayed however three weeks due to administrative issues in accessing the virtual machine provided by the university.

The period of collection was from June 22 at 5:40 pm until July 13 at 10:50 am. To access the HERE API a class of Python functions were created to generate data requests and to collect the correct geographic subset of data for Oxford [1] [2]. Data was collected using a Python script [3] which was triggered every five minutes by a cron job. A cron job is a file that contains a schedule of procedures to run at specific times, which is built into linux systems. The data was saved in CSV format with the following information: road name, direction, latitudes, longitudes, speed (capped by limit), speed (uncapped by limit), free flow speed, and jam factor. Speed capped by limit was the variable of interest because minimum speeds are a concern and having values that are higher than the speed limit may inflate the model predictions.

5.1.4 Weather Data Description

Mentioned previously there has been research in the field of traffic prediction incorporating weather data, which although is a complex task was applied. Sources for weather data in Oxford vary, however much of the local weather observations are collected hourly, when the target would be to have sub-hourly observations. Tomorrow.io (Tomorrow.io, 2021) is a weather solutions company that provide an API that contains the current weather conditions for a geographic location. This service is updated every 15 minutes, which satisfies the criterion of sub-hourly data. In addition, they offer a variety of weather variables. Essien *et al.* (2021) used weather observations of temperature, humidity, rainfall, cloud cover, and a wide range of other variables when developing their model. For this project the following variables were collected: precipitation intensity, precipitation type, wind speed, wind gust, temperature, humidity, cloud cover, and weather condition.

Data was collected over the same period as the traffic data, except on 15-minute intervals. Tomorrow.io provided a base Google script which collects weather variables each time the script is executed (LironRS, 2021). This script was modified and set running on Google Sheets every 15-minutes, which saved the data in CSV format for the period of collection [5].

5.2 Analysis

5.2.1 Initial Traffic Data Analysis

The average speeds of the Peartree roundabout range between 40 to 100 km/h which can be seen in Figure 11. The largest changes in speed occur on the A34, which is understood since this is the fastest road in the locality, and during rush hour traffic speeds can slow dramatically.

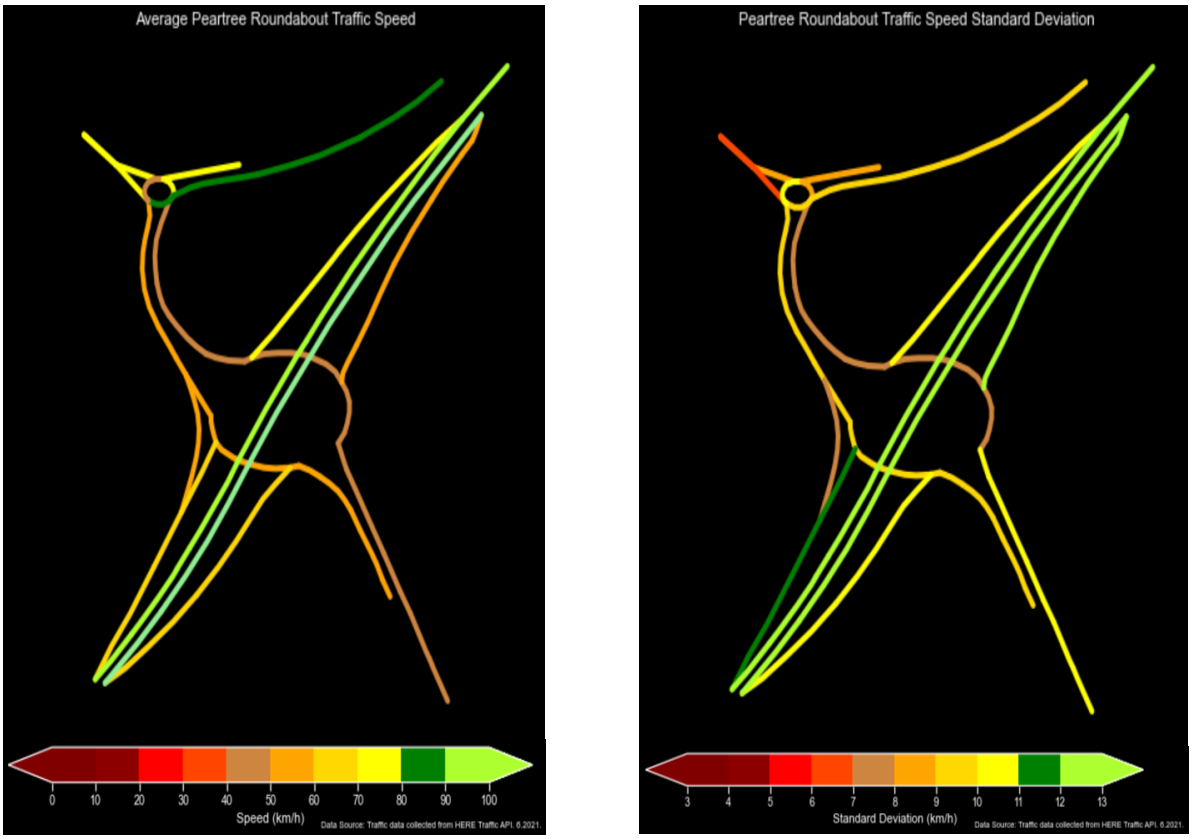


Figure 11: Visualizations of the Peartree Roundabout average speeds (left) and the standard deviation of those speeds (right).

By averaging the speeds across each of the 70 road segments, the average speed of the entire locality can be captured across varying time periods in Figure 12. There is a clear diurnal cycle in road speeds, with sharp troughs that occur on Friday evenings during rush hour. Average speeds rise overnight during hours of low traffic density and decrease sharply in the morning commute hours on weekdays.

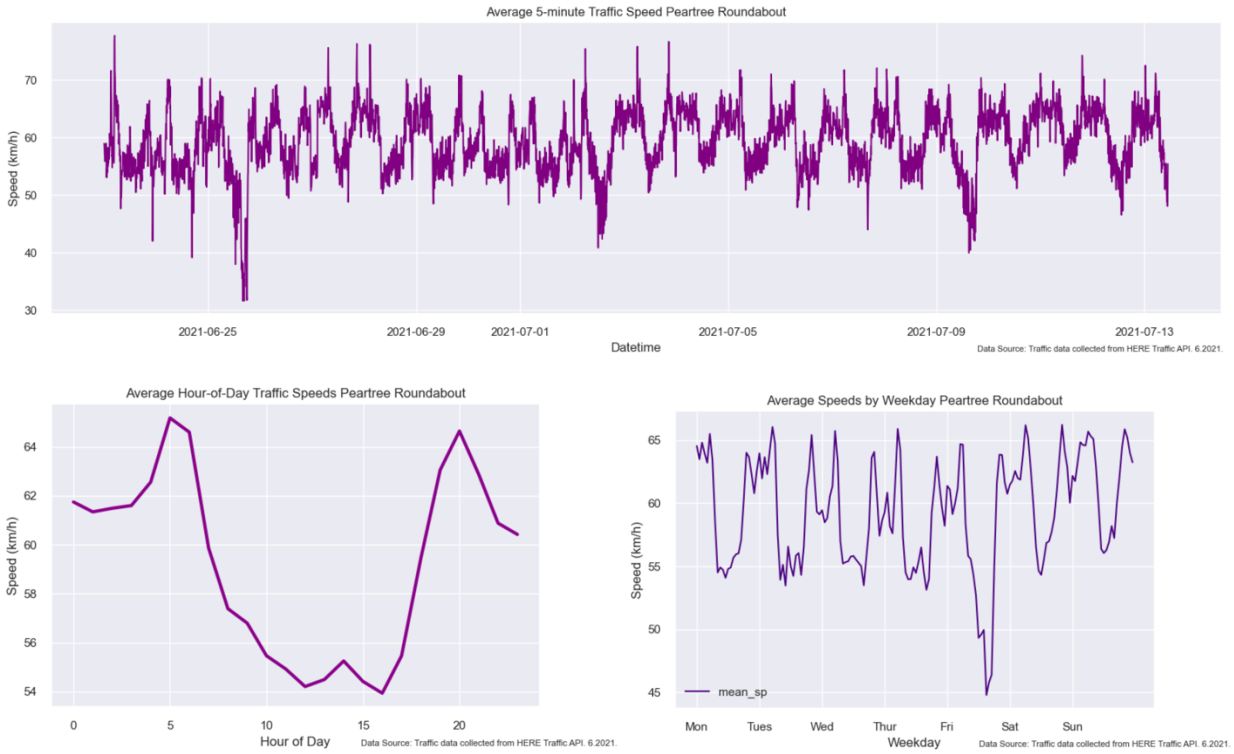
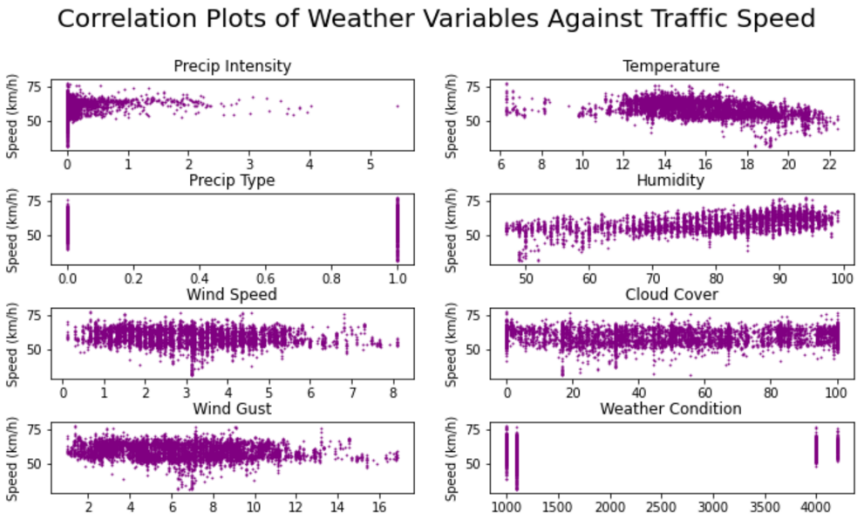


Figure 12: Average speeds across the entire roundabout by varying time periods. Entire period of data collection (top), by hour of the day (bottom-left), and by weekday (bottom-right).

5.2.2 Initial Weather Data Analysis

Linear regression was applied as a dimensionality reduction to the weather variables with the goal of reducing the number of parameters. Using a p-value of 0.05 as the cutoff for significance, only the 4 variables of precipitation intensity, temperature, humidity, and weather condition remained significant. A correlation plot (Figure 13) of all the weather variables against traffic speed also confirms this, with the most outright influential predictor being humidity with a slight positive correlation.



OLS Regression Results

Dep. Variable:	sp	R-squared:	0.332
Model:	OLS	Adj. R-squared:	0.331
Method:	Least Squares	F-statistic:	730.8
Date:	Thu, 02 Sep 2021	Prob (F-statistic):	0.00
Time:	17:18:54	Log-Likelihood:	-17012.
No. Observations:	5891	AIC:	3.403e+04
Df Residuals:	5886	BIC:	3.407e+04
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	42.9857	1.047	41.074	0.000	40.934	45.037
precipitationIntensity	0.4403	0.194	2.265	0.024	0.059	0.821
temperature	-0.1139	0.034	-3.349	0.001	-0.181	-0.047
humidity	0.2134	0.007	29.830	0.000	0.199	0.227
weatherCode	0.0004	5.22e-05	7.538	0.000	0.000	0.000

Omnibus:	146.263	Durbin-Watson:	0.230
Prob(Omnibus):	0.000	Jarque-Bera (JB):	206.766
Skew:	-0.277	Prob(JB):	1.26e-45
Kurtosis:	3.732	Cond. No.	4.04e+04

Figure 13: Correlation plots of weather variables before dimensionality reduction (top). Results of removing insignificant variables (bottom).

Although there are four variables of significance, it is worth noting that during times of high humidity there is usually a form of precipitation which would indicate that both humidity and precipitation intensity may be correlated. In addition, temperature may be a significant feature because it follows the same diurnal cycle as traffic speeds: large changes in the morning and afternoon hours with peaks and troughs overnight and midday. Based on this evidence it would make sense to eliminate both temperature and humidity, however, due to the small amount of data that has been collected removal of these variables cannot be justified.

5.3 Graph Representation of the Peartree Roundabout

To best represent the spatial domain in non-Euclidean space, the literature stated that graphs are the best method. Therefore, the collected traffic data was converted into a graph structure, using the Python library NetworkX (Hagberg and Swart, 2008) to build the graph manually. This process required determining which road segments (nodes) connect to adjacent neighboring road segments in the roundabout to form an undirected graph. An undirected graph was represented because the traffic in a road segment ahead can impact the traffic at the current road segment, and conversely the traffic at a previous road segment can impact the traffic at the current road segment. The graph (Figure 14) and adjacency matrix were produced, which is necessary for modelling the traffic data.

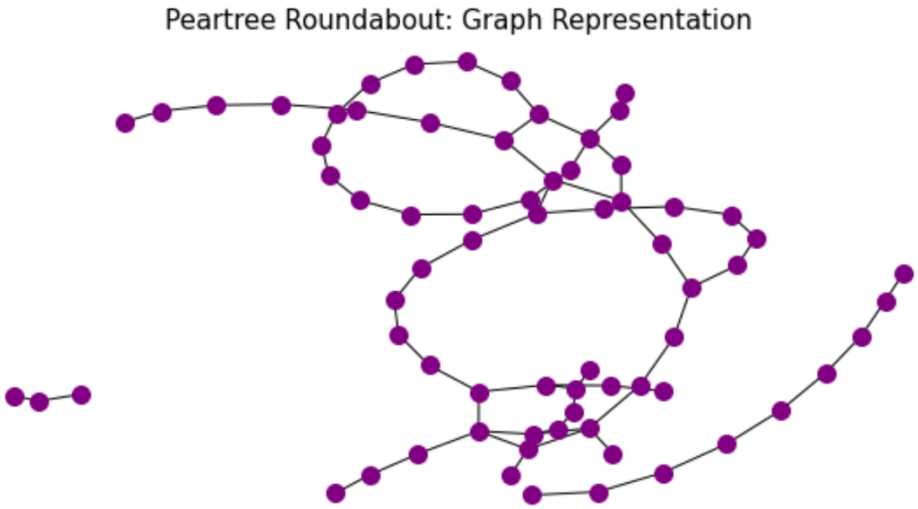


Figure 14: Graph representation of the Peartree Roundabout.

6. Methodology

A hybrid deep neural network was employed to make short-term traffic predictions across a variety of prediction horizons using both traffic and weather data. This complex architecture (T-GCN-wx) was compared against three other modeling approaches, linear regression, LSTM, and T-GCN to critically analyze the candidate model. Only the

T-GCN-wx method involved weather and traffic data, with the other remaining three only using traffic data alone. This allowed for a fair comparison against the effectiveness of weather data. In this section first the workflow of the project will be presented, followed by a discussion of the architectures of the candidate model and the three benchmark methods, along with justifications of the methods, the training and testing of each model, hyperparameters, and the metrics that will be used in the analysis.

6.1 Workflow

6.2.1 Tasks

Several tasks were required to bring the aim of this research to fruition, including the following:

- **Data Access:** Gaining access to both the traffic and weather data source API's.
- **Data Collection Mediums:** Writing the data collection scripts in Python and Google scripting language.
- **Data Collection and Storage:** Running the scripts for a three-week period to collect and store the data on a virtual machine.
- **Preliminary Data Analysis:** Analysis of the raw data and preprocessing to remove outliers, null values, and to temporally correct the weather data.
- **Model Development:** Developing the models necessary for the research in Python using the necessary Python packages.
- **Hyperparameter Tuning:** Tuning of the hyperparameters of the models.
- **Model Training and Testing:** Training and testing of the models using Jupyter Notebooks with varying prediction horizons.
- **Final Analysis:** Completing a comprehensive analysis on the candidate model in Python using Tableau and seaborn to visualize the results. The analysis should assess the temporal, spatial, and overall performances of the candidate model against the benchmark methods.

6.2.2 Gantt Chart

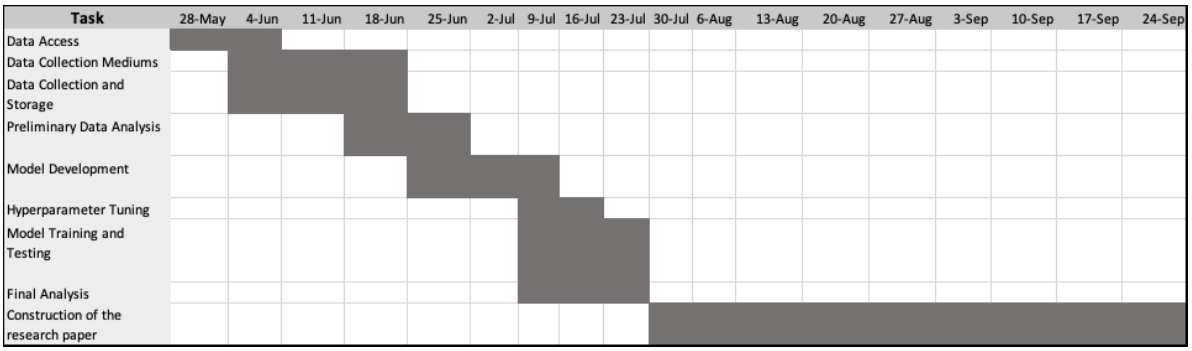


Figure 15: Gantt chart of the tasks necessary to complete the dissertation.

The Gantt chart in Figure 15 shows the timeline of tasks that were required to complete the dissertation. The most time-consuming tasks involve both the data collection phase and the construction of the final paper. This chart is the estimates before any of the tasks were completed and will be revisited again in the conclusion.

6.2 Machine Learning Problem

6.2.1 Data Preprocessing

The machine learning problem is a supervised problem, meaning there is a target value that can be predicted. Data was split into training and testing, using an 80/20 split which is a standard ratio for supervised learning problems. Training and testing data were then reorganized into sequential inputs of one hour in length [4], and the target variable was established as the N number of steps into the future from the current timestep (t_0) (Equation 7). This is otherwise known as the prediction horizon.

$$[X_{t-1}, X_{t-2}, X_{t-3}, X_{t-4}, X_{t-5}, X_{t-6}] \rightarrow X_{t+N}$$

Input sequence Target

Equation 7: The equation of the time series traffic prediction problem, with N being the number of steps into the future to predict.

This produced a multidimensional array for each input sequence, and could then be incorporated alongside a machine learning algorithm of choice. For this research an input sequence of one hour (12 five-minute data points) was used and a prediction horizon of 5, 15, 30, and 60 minutes were considered. This is similar to both Zhao *et al.* (2019) and Yu, Yin, and Zhu (2018) who used one-hour inputs to forecast 15, 30, 45, and 60 minutes into the future.

Weather data was also processed similarly, with weather variables being organized into sequential inputs. In addition, weather data was linearly interpolated from a 15-minute time scale to a 5-minute time scale for compatibility with the traffic data. The target variables however remained the traffic speeds.

6.2.2 Normalization

Normalizing is a standard machine learning procedure which ensures that features are on a universal scale, and this allows machine learning algorithms to train faster and for gradient descent to achieve global minima faster (Samit, 2019). For both the traffic and weather data, maximum minimum normalization between 0 and 1 was performed (Equation 8) [4].

$$A' = \frac{A - A_{min}}{A_{max} - A_{min}}$$

Equation 8: Max min normalization formula which converts values between 0 and 1 (Patro and Sahu, 2015).

6.3 Model Architectures

One candidate architecture was developed for this research, which is discussed first, followed by three other architectures which are used for critical comparisons only.

6.3.1 T-GCN-wx

T-GCN Concept

The T-GCN-wx is the candidate model and focus for this research, as it incorporates both traffic and weather data using a deep hybrid neural network structure. This structure is heavily based on the T-GCN architecture developed by Zhao *et al.* (2019) which involved using a GCN to process the graph structure of the road network, and a GRU to process the temporal features of the data. Stellargraph (Data61, 2018) is a Python library that provides a built-in function which is inspired off of the paper by Zhao, using a GCN combined with an LSTM to make traffic predictions. As mentioned above, LSTM and GRU have similar performance and therefore and using LSTM would not pose a performance issue.

The structure of a 1-layer GCN can be represented as the following

$$f(X, A) = \sigma(\hat{A}\, ReLU\big(\hat{A}XW_0\big)W_1)$$

- $\hat{A}$: This is a preprocessing step $\hat{A} = \widetilde{D}^{-\frac{1}{2}}\tilde{A}\widetilde{D}^{-\frac{1}{2}}$.
- W_0 : This is the weight matrix $W_0 \in R^{P\ x\ H}$; P feature length, H is the number of hidden units.
- W_1 : This is the weight matrix $W_1 \in R^{H\ x\ T}$; T is prediction length.
- $F(X, A)$: This is the output with prediction length T.
- $ReLU$: This is the activation function for the GCN layer, which can be changed.

Equation 9: The equation of a 1-layer GCN (Zhao et al., 2019).

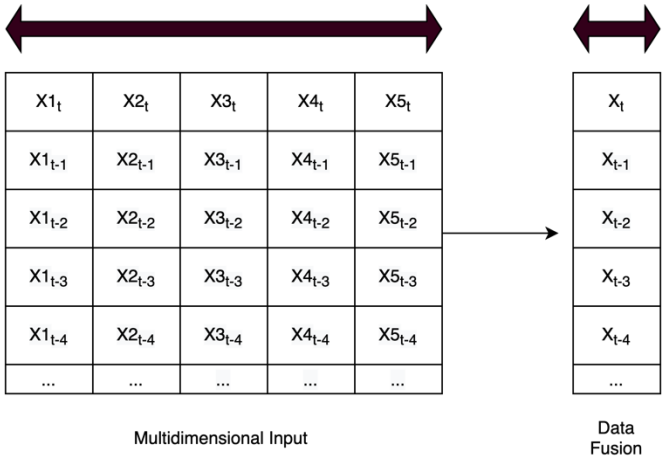
In addition to the feature matrix being input to the GCN layer, the adjacency matrix also must be input for the GCN to understand the connections in the temporal graph. The output of the GCN layers then passes to the LSTM, where it is stored in both short and

long-term memory as it propagates though each cell. The result produces a traffic prediction *N* steps into the future.

Incorporating Weather Data

A method of data fusion was implemented to incorporate weather data. A review of data fusion techniques resulted in choosing a data-in data-out (DAI DAO) data fusion technique (Castanedo, 2013). This was also adopted by Essien *et al.* (2021) where a traffic model combined weather variables and traffic speeds in the first layer of a deep neural network. Using a multidimensional input, the traffic data and weather variables were fused in the second layer. This tends to lead to more reliable outputs as the errors introduced at the prediction level are avoided.

The traffic data and weather data were combined into a four-dimensional input matrix, with the extra dimension being the five features (four weather features one traffic feature). Using an input layer with five dimensions, this multidimensional array can be fed into a neural network. The second layer reduces the five dimensions to one, hence "fusing" the data which can be better represented in Figure 16.



- $X1$: The traffic speeds.
- $X2 - X5$: The weather features.
- t : Initial time.
- X : The weather and traffic fused data.

Figure 16: A diagram of data fusion process between weather and traffic variables.

Model Architecture

The complete model architecture is visualized in Figure 17 which includes the data fusion layer and the complete T-GCN. Data enters the model with four dimensions: time, number of nodes, input sequence length, and the number of features. The data is dimensionally reduced after passing through a dense layer of neurons, and batch normalization and a dropout layer are applied to prepare the data to be fed into the T-GCN. At this phase, the traffic data is fused together with the weather data. The adjacency matrix also is input into the T-GCN model at this phase. Upon entering the T-GCN, the data passes through a GCN layer which spatially filters the data before it is reshaped to enter the LSTM layer. After the LSTM layer the data passes through a dropout layer and one more neuron layer before the final forecast is produced.

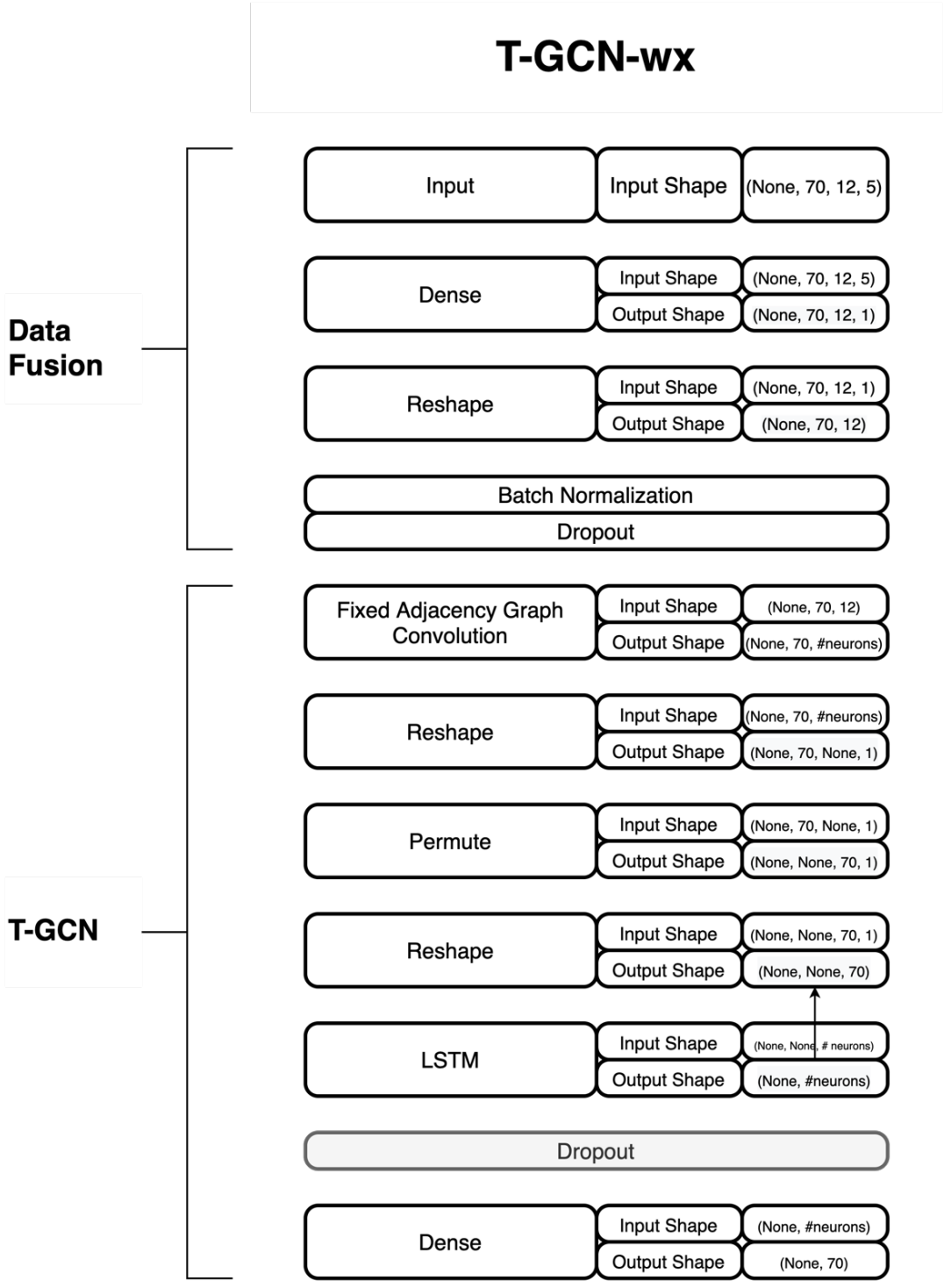


Figure 17: A complete overview of the T-GCN-wx architecture.

6.3.2 T-GCN

The T-GCN model without using weather data also was considered for this study, mainly for comparison purposes. To determine the impact that weather data has on the traffic prediction problem, a comparison against an identical model without weather data would allow for critical analysis. The T-GCN does not include a data fusion layer, and simply includes the GCN and LSTM cells for spatiotemporal modeling (Figure 18).

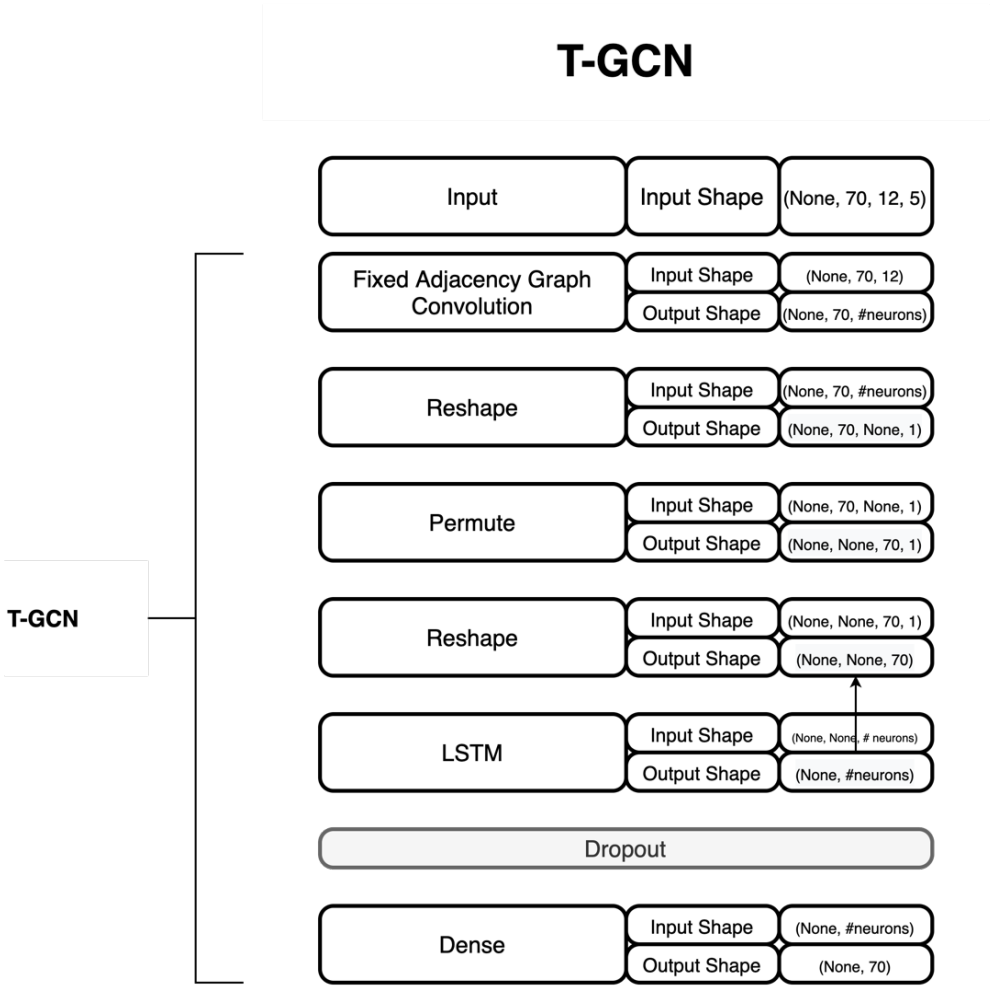


Figure 18: The T-GCN model architecture.

6.3.3 Linear Regression

Linear regression is both simple to implement and has no hyperparameters to tune which makes it an excellent benchmark method to compare against complex models. In traffic prediction linear regression is rarely used because of its inability to define non-linear relationships, which is a major characteristic of traffic prediction. Linear regression also does not account for the spatial variability in the traffic network. However, for the purposes of this experiment this method will be employed. Weather data was not incorporated into this model, as the focus of the linear regression was to create a benchmark performance for the traffic data alone.

The road network data was organized into training and testing three-dimensional arrays, with columns being the 70 road segments, rows being the traffic speeds on a 5-minute interval, and the third dimension being the number of 1-hour input sequences for training. The target values were the traffic speeds at times 5, 15, 30, and 60 minutes into the future. Using the scikit-learn library (Lars *et al.*, 2011), a linear regression algorithm was trained on each road segment and predictions made for each node accordingly.

6.3.4 LSTM

LSTM is a common solution to the traffic prediction problem; its implementation is relatively simple because of the simple syntax of the library keras. An LSTM network was implemented, using the same input matrix as was used for the linear regression. This is because the LSTM cell cannot understand a spatial graph structure, and therefore the data must be passed in matrix form. Figure 19 is an overview of the LSTM network architecture, which is a simple design. The purpose of the LSTM is to make comparisons against the T-GCN design and determine if the spatial component makes a significant contribution to the overall performance of the model.

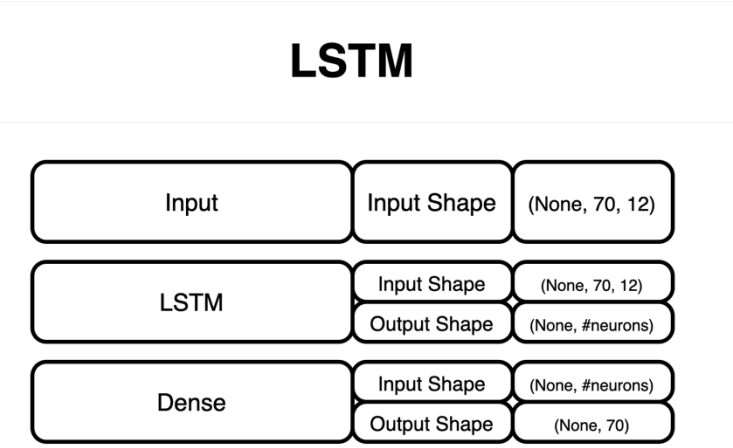


Figure 19: The LSTM network architecture.

6.4 Hyperparameter Tuning

For the T-GCN-wx, T-GCN, and LSTM models there were hyperparameters that require optimization. This included the learning rate, activation function, number of layers, number of neurons in each layer, batch size, and many others. This was a time-consuming process, and for the scope of this study hyperparameter tuning was performed on the T-GCN and T-GCN-wx only, with the activation function and number of neurons in each layer being optimized. The number of layers for both the T-GCN, T-GCN-wx, and LSTM were all limited to 1. It is worth noting that the T-GCN and the T-GCN-wx are hybrid models and both will have multiple layers, however the number of layers for each specific algorithm were also limited to 1.

For hyperparameter tuning, the number of epochs set was 20 to determine which method had the potential to perform best. Typically, a much large number for epochs is used, however due to the computer limitations the number had to be set very low to get timely results. The learning rate also remained set at 0.001 as standard practice with the optimizer “adam”. Given that this machine learning problem is non-categorical, the loss function used was mean-squared error (MSE). The reasoning for this is mean-squared error is a typical metric for regression-based prediction. K-fold cross validation

was not considered as it cannot be easily applied to time series prediction, and therefore was not considered in the hyperparameter tuning process.

6.5 Model Training and Testing

A varying number of epochs were used for the training of the LSTM, T-GCN, and T-GCN-wx networks. Models were trained until the point before overfitting began to ensure the highest performance. Overall, the number of epochs necessary to reach optimal performance decreased as the prediction horizon increased. The number of epochs between models and prediction times lay between 30 and 100 epochs. Training time also varied significantly between models, with the T-GCN-wx requiring the longest to train in comparison to the LSTM and linear regression models. All model training and hyperparameter tuning was carried out in Jupyter notebooks, however an excerpt of this code was written to a Python file for training the models on 5-minute prediction lengths. This example Python code is available in the code appendix [6]. Training on all remaining prediction lengths followed a similar procedure and code structure.

The training period of data was from June 23 at 5:40 pm - July 9, 2021 at 9:35 am, and the testing period was from July 9 at 9:40 am - July 13, 2021 at 10:40 am. Altogether, 16 days of training data and 4 days of testing data.

6.6 Evaluation Metrics

Nearly all research comparing traffic prediction methods use multiple metrics for model comparison including mean absolute error (MAE), R2 score (R2), accuracy, variance, and root mean squared error (Zhao *et al.*, 2019; Sun, Wu and Xiang, 2020; Essien *et al.*, 2021). For this study MSE, MAE, and R2 were employed given their widespread usage and because MSE is the loss function. The three metrics can be written as the equations in Equation 10:

$$MSE = \frac{1}{n}\sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)^2$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n} | Y_i - \widehat{Y}_i |$$

$$R^2 = 1 - \frac{\sum_{j=1}^{M}\sum_{i=1}^{N}(y_i^j - \widehat{y_i^j})^2}{\sum_{j=1}^{M}\sum_{i=1}^{N}(y_i^j - \bar{Y})^2}$$

Equation 10: Metrics for evaluation equations (Zhao et al., 2019; Sun, Wu and Xiang, 2020).

For all metrics the overall performance of the model is equal to the average value of the metric across the 70 road segments. For example, the resulting MSE for the T-GCN model would be the average MSE across all road segments and forecast time steps.

6.7 Constraints and Limitations

6.7.1 Available Data

Hourly or sub-hourly traffic data is unavailable for Oxford city that is of public use. The Oxfordshire City Council do claim to collect data using in-situ traffic sensors for Oxford city, however no response was received from the Council upon a formal request for the data (Council). Therefore, data was collected from a commercial traffic provider.

6.7.2 Available Storage

The amount of traffic and weather data that was collected was only three weeks, which is significantly less than what is standard practice for traffic modeling. Additional data also would have created a data storage problem, as over 14 GB of data was collected only for three weeks.

6.7.3 Computational Resources

The entirety of the training and testing of the model was performed on a personal laptop, which is not optimal for machine learning tasks. Therefore, the extent of

hyperparameter tuning was cut at the expense of the time it took to complete a computational task.

6.8 Legal and Environmental Considerations

Only legal and environmental considerations can be considered in the scope of this dissertation, given that this study does not in any way impact individuals ethically or socially.

6.8.1 Legal

Traffic and weather data was collected from both HERE and Tomorrrow.io legally from their API's. Both have a policy which permits for public use of their data, in addition emails were received from both companies confirming that their data may be used for this specific dissertation.

6.8.2 Environmental

Three months of this dissertation involved data collection, modeling, and analysis on a computer which consumes electricity. Even though the impact is nearly negligible, the usage of electrical energy in this project was quite high. The environmental impact of producing energy for consumption is detrimental to the environment, and as research into deep learning continues the environmental impact should also be considered.

7. Results and Analysis

Results were gathered from the candidate model and the three benchmark models for critical analysis. Input sequences for all models were 1-hour and prediction lengths were 5, 15, 30, and 60-minutes into the future. Visualizations were created using matplotlib, seaborn, and Tableau. Statistical significance was determined as having a p-value less than 0.05.

7.1 Overall Performance

Table 2, Table 3, Table 4, and Table 5 all include detailed performance metrics for all four models and the prediction lengths. Overall results indicate that the T-GCN and T-GCN-wx models outperformed the linear regression and LSTM models in most metrics. Based on MSE and R2, the T-GCN slightly outperformed the T-GCN-wx overall, however the difference is insignificant (Figure 20).

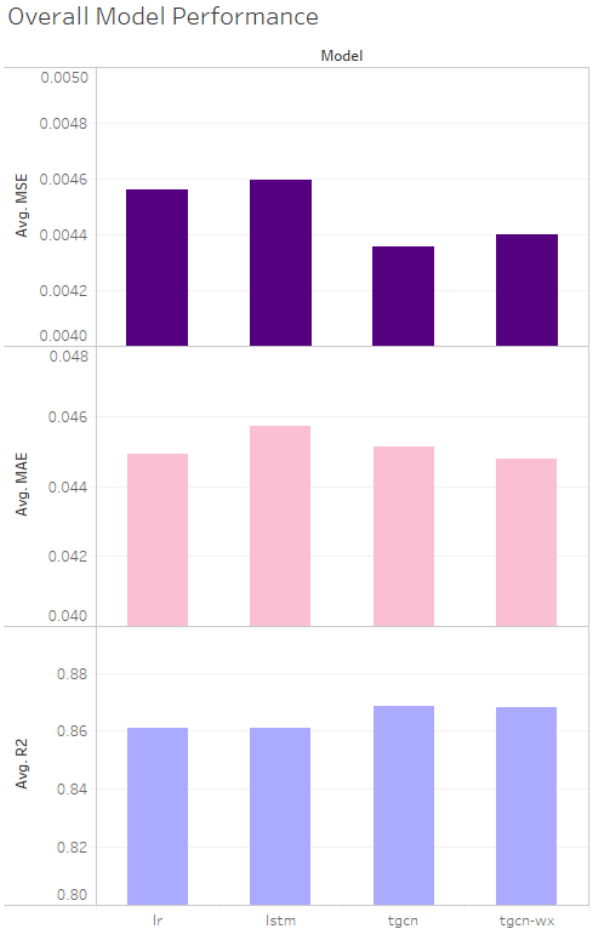


Figure 20: Overall model performances by metric.

As prediction length increased, the deep learning models performed increasingly better than the linear regression and LSTM models (Figure 21). It is worth noting however for the first prediction length of 5-minutes, the linear regression performed on par with the T-GCN. This is likely because prediction requires increasing insight the larger the number of steps into the future that is being predicted, and one step into the future

requires very little insight. At a prediction length of 60-minutes, the linear regression is the worst performing model whilst the graph-based models are the best performing.

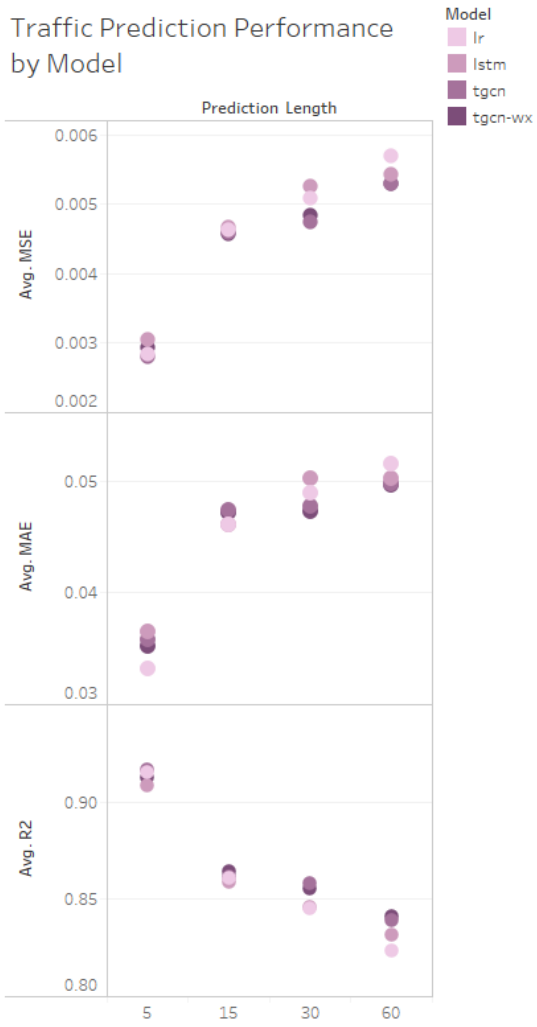


Figure 21: Model performance by metric and prediction length. As prediction length increases, the disparity in performance also increases between models.

7.2 Statistical Significance

Using a t-test for statistical significance and a p-value of 0.05 for significance, comparisons between the candidate model (T-GCN-wx) and the other benchmark methods were made. Results indicated that only in terms of MAE, the T-GCN-wx

performed better than the linear regression and the LSTM for prediction times greater than 5-minutes. For MSE and R2, all model performances were statistically insignificant from one another, which would indicate that all the models were equal. However, this must be interpreted with the understanding that very little data was used, and therefore the differences between the models are very small.

7.3 Temporal Performance

Due to small errors between the models, very small differences exist on the temporal scale. The They can be pinpointed to specific times of the day when traffic speeds are changing rapidly from heavy rush hour traffic. Figure 22 shows the largest error between 1:00 pm to 7:00 pm, with peaks around 4:00 pm - 6:00 pm which is rush hour. During these periods the T-GCN-wx and T-GCN were the overall best performers. Weekend versus weekday comparisons indicate that all models perform over 60% better on the weekend despite the relatively small amount of weekend training data that was available Figure 23. This can be attributed to weekend traffic volatility being significantly less than weekday traffic. Comparisons between models for weekend versus weekday performance are insignificant.

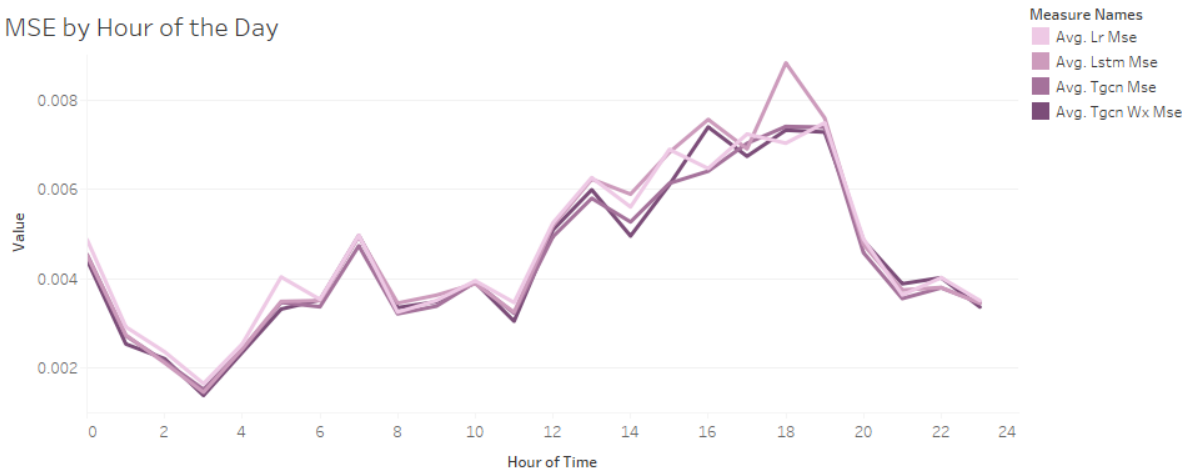


Figure 22: Model MSE by hour of the day. Error increases in all models in the evening rush hours, with the deep networks performing best during these times.

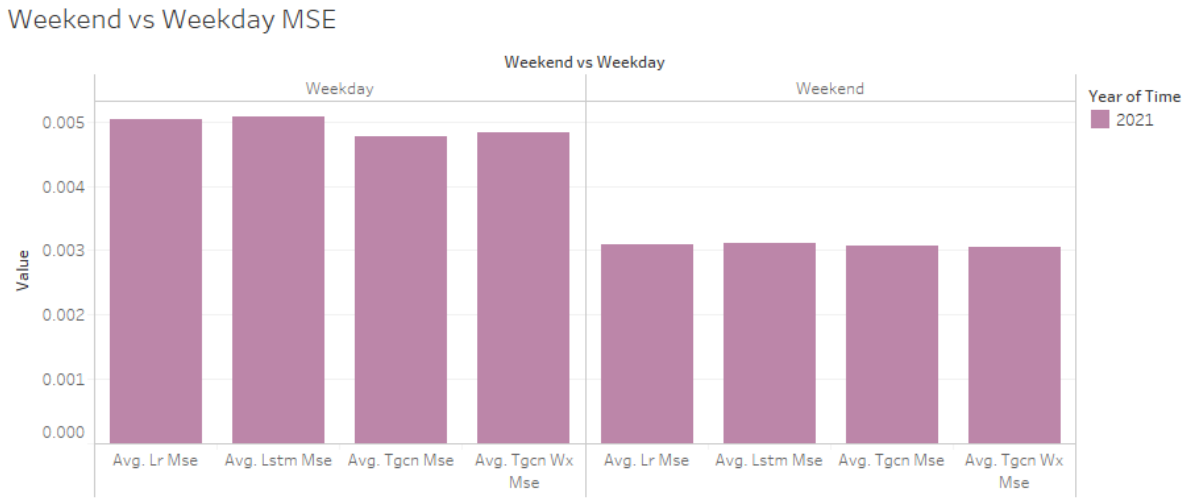


Figure 23: Model comparisons of MSE between weekend and weekday traffic prediction.

7.4 Spatial Performance

Overall spatial performance differences between models were negligible. The maps of the road network in Figure 24 show nearly identical performance for the T-GCN, T-GCN-wx, and the linear regression model. The graph-based spatiotemporal models only performed with 4.3% less MSE than the non-graph models, which is very little

improvement. Visual performance charts would suggest that the improvements have little to do with the spatial representations of the graph-based models. The highest MSE values on the visualizations suggest that the A34 is not well predicted, and this leads to poor performance on surrounding on-ramp and off-ramp connections.

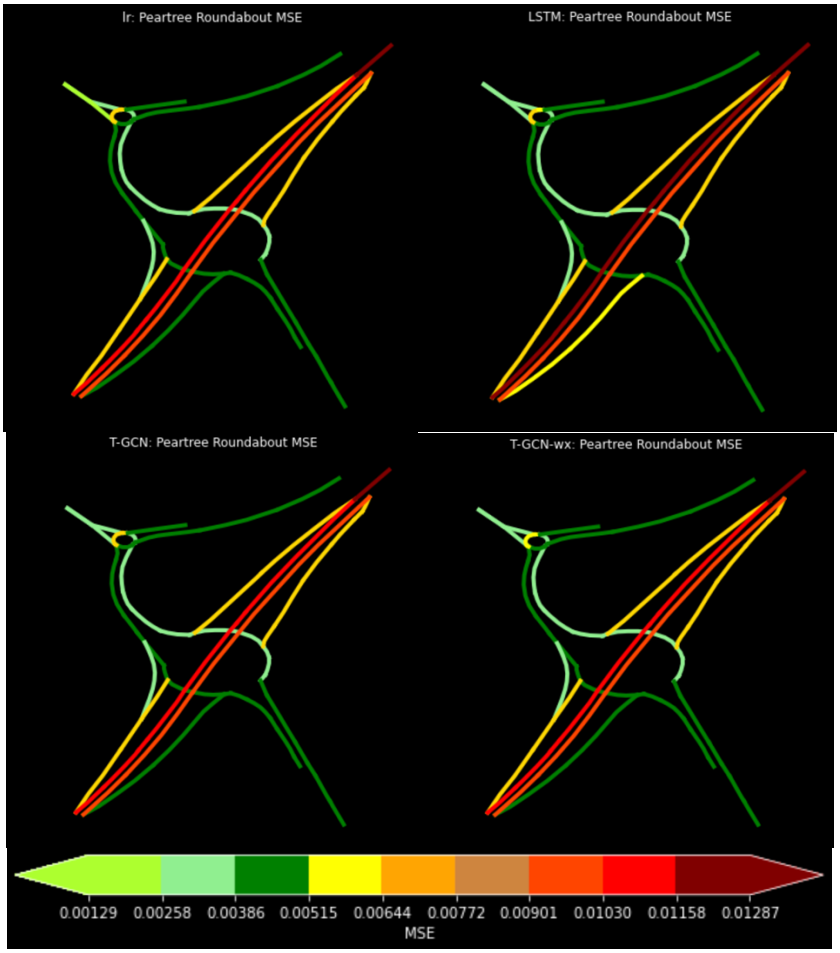


Figure 24: Spatial MSE by Linear Regression (top left), LSTM (top right), T-GCN (bottom left), T-GCN-wx (bottom right).

7.5 Candidate Model Performance

7.5.1 T-GCN-wx Performance

The T-GCN-wx performed better than the non-graph models after a time prediction length greater than 5-minutes. In comparison with the T-GCN the model across all metrics and prediction lengths, the differences in performance are +-2% which are

negligible. A prediction lag is evident in Figure 25, which eventually dissipates as the prediction length increases.

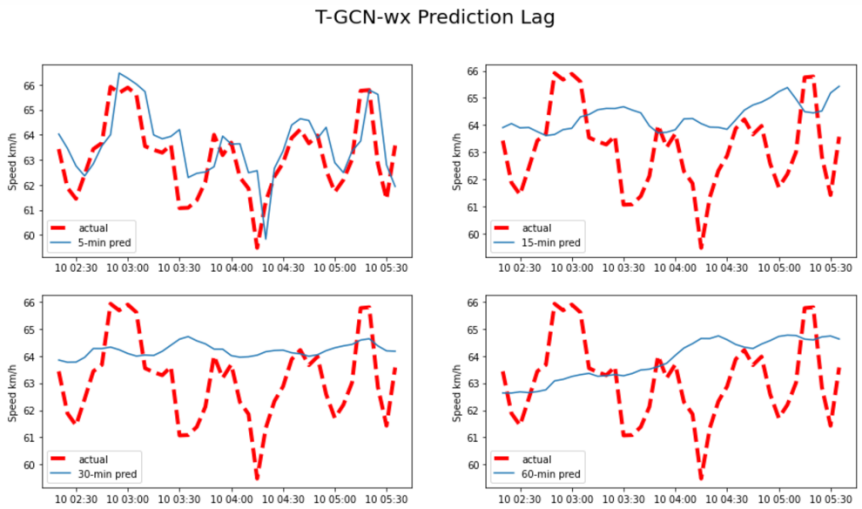


Figure 25: T-GCN-wx performance by prediction length. There is an evident lag that decreases as prediction length increases.

7.5.2 Weather Impact

The impact of weather variables in the T-GCN-wx model can be summarized as statistically insignificant from the previous comparison with the T-GCN model performance. A more in depth look at performance can be found in Figure 26 and Figure 27, where the T-GCN-wx performs better during periods of sun, rain, and partly cloudy conditions. However, due to a lack of sufficient weather variables across all categories (particularly rain), these performances should be disregarded until more data can be collected and tested.

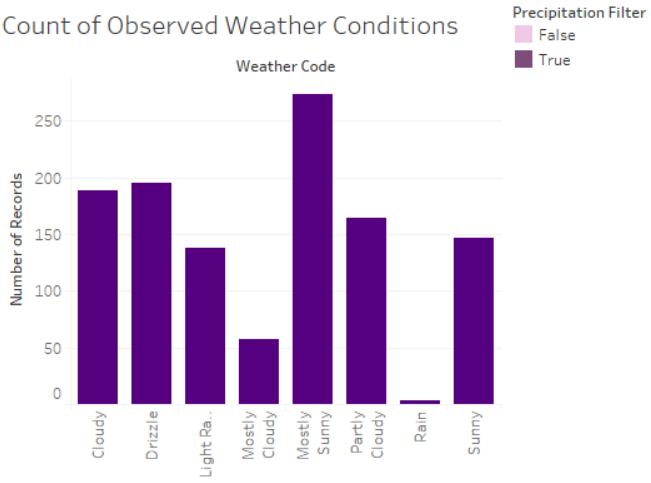


Figure 26: Count of all weather conditions collected.

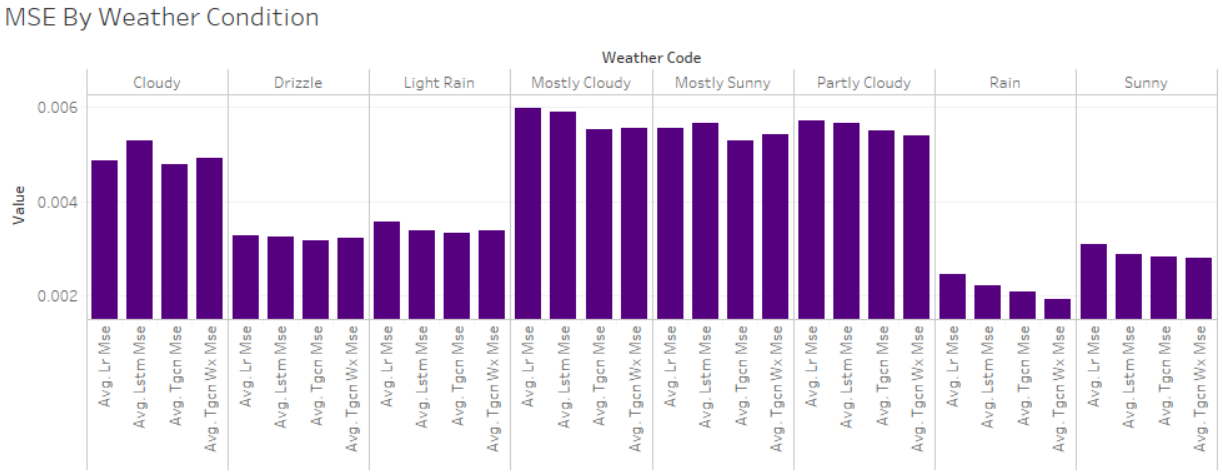


Figure 27: Model performances during specified weather conditions.

Figure 28 is a pairplot which is color shaded to represent times when the T-GCN-wx model was the best performing model. The diagonal plots indicate nearly identical distributions between times when the T-GCN-wx was the best performer and when it was not for each weather variable. This would suggest that the T-GCN-wx does not perform better or worse under any weather conditions.

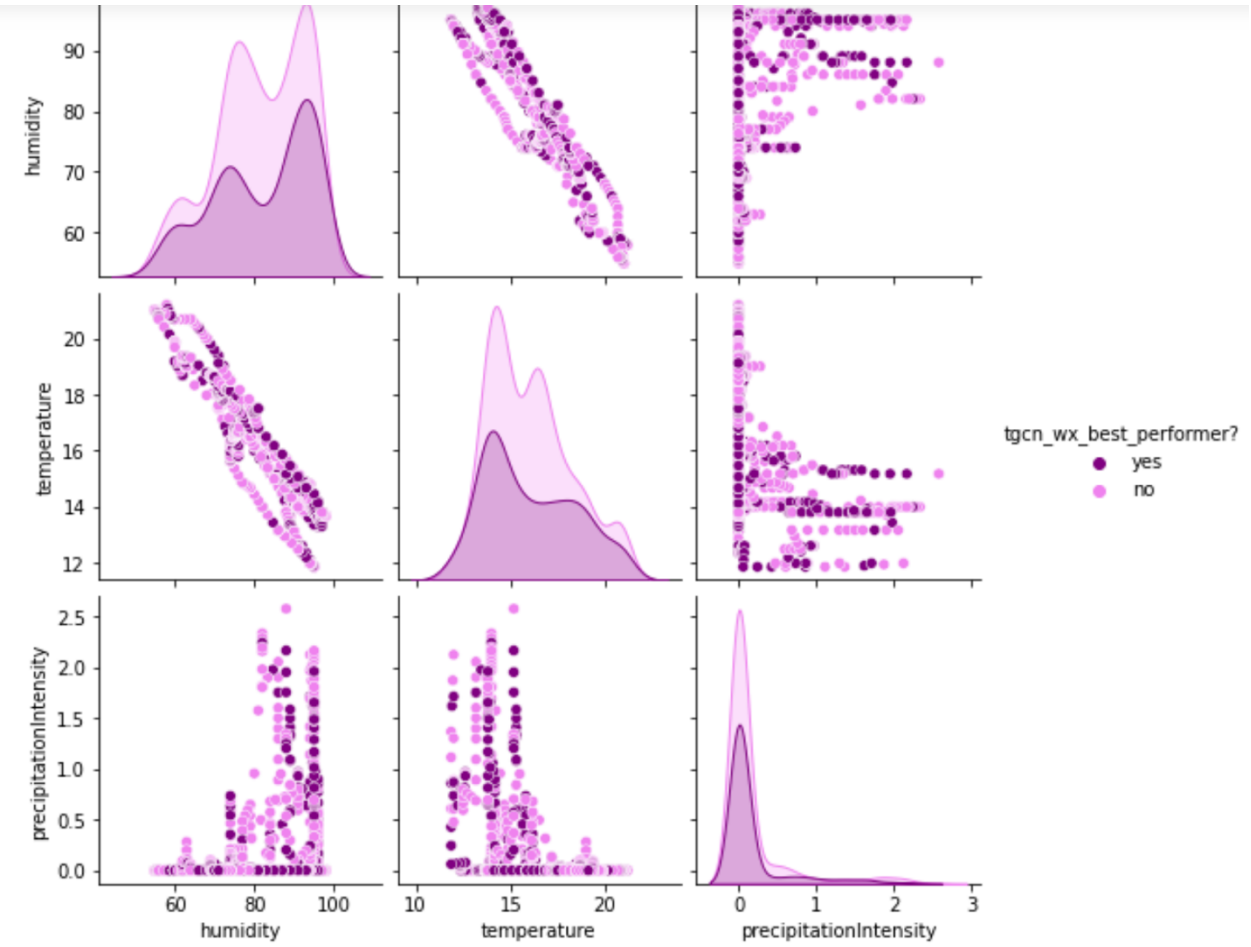


Figure 28: T-GCN performance by weather variable. Dark purple shadings represent times when the T-GCN-wx was the best performing model, and pink represents the opposite. The diagonal plots indicate that the distributions between times when the T-GCN-wx was the best model and times when it was not are nearly identical, suggesting that the model does not perform better during certain weather conditions.

8. Discussion and Conclusion

8.1 Discussion

A short-term traffic deep neural network model for Oxford city was proposed, developed, and tested using commercial traffic and weather data that was collected over a three-week period. The candidate model (T-GCN-wx) was developed and tested alongside three benchmark models (linear regression, LSTM, T-GCN), to make critical comparisons between certain features of the candidate model. Each model was

hyperparameter tuned, trained, and tested using 1-hour of inputs to forecast traffic 5, 15, 30, and 60-minutes into the future. The linear regression model is a basic statistical model that was implemented to gauge the effectiveness between neural networks and regression for traffic prediction. The LSTM is a non-spatial but temporal model which allows for comparisons against the spatial components of the candidate model. The T-GCN model is both spatial and temporal but does not contain weather data and therefore can be compared against the candidate model to gauge the effectiveness of weather data.

Overall performances of the models suggest that the candidate model performed on par with the T-GCN model. For 5-minute traffic prediction, the linear regression and T-GCN were the best models, which suggests that at low prediction lengths simple statistical models can make good predictions on par with deep neural networks. However, once prediction lengths increased the deep neural networks performed best, which is what was expected. Using a t-test for statistical significance, it was found that very few statistical differences exist between models, however due to the small dataset that these models were trained on this could be expected.

The temporal performance of the model highlights slight differences between the deep learning graph-based methods and the non-graph methods. During periods of high traffic, the graph-based models performed best. In addition, graph-based methods performed better on both weekend and weekday times, although these numbers require more data to fully test the performance. Overall performance indicates that very little overall differences exist between model performance on the temporal domain (less than 2% difference), with the linear regression model performing well on short prediction horizons.

The graph representation of the road network resulted in little improved performance over the non-graph models, and very little visual improvement on the road network. This would suggest that the graph representation did very little to improve model performance, which was not expected. A reason for this could be that typically graph-

based traffic prediction uses much larger roadways which span multiple neighborhoods or even cities which can impact the long-term traffic prediction. In this study, only the Peartree Roundabout was used which constitutes a very small sub-neighborhood size road network. In addition, the poor performance on the A34 highway can be understood since there are no connecting nodes, or A34 road segments, to the A34 freeway. Therefore, any traffic from other road segments of the A34 cannot be accurately captured by the model and therefore these errors are propagated throughout the road network. A solution to this would be to include more sections of the A34 which would provide the models with a better understanding of the road network and hopefully improve performance.

The impact of weather variables on traffic prediction was made by comparing the T-GCN with the T-GCN-wx. No differences in prediction existed for the weather variables humidity, temperature, and precipitation intensity. For weather conditions, a few conditions where the candidate model performed best were identified, however the amount of data necessary to make concrete conclusions was not available and therefore there is no evidence to suggest that weather variables improved the predictive capabilities of the candidate model.

8.1.1 Workflow Discussion

The Gantt chart that was created in Figure 15 severely underestimated the issues that were encountered in the data collection phase. There were several issues with accessing the University virtual machine to collect data, specifically administrative issues. These were eventually worked out by mid-July, however this phase of data collection was twice as long as expected. In addition, hyperparameter tuning was a three-week process due to computational deficiencies of the researcher's laptop. Altogether, this delayed the remainder of the tasks necessary to complete the dissertation and resulted in the construction of the final paper to occur in late August.

8.1.2 Discussion on Administrative, Environmental, and Legal Issues

Administrative issues arose during the data collection phase and delayed the project by an additional two weeks. These minor bumps were not accounted for at the beginning of the dissertation and the process of mitigating the issue required consulting with the University IT department. Environmental issues regarding significant use of energy whilst collecting, storing, and modelling data were expected although they can be assumed to be negligible for the short period of time this research encompassed. Finally legal issues were mitigated confirming with each commercial data provider that their data be used for research purposes.

8.2 Conclusion

This project proved a concept and proposed a model architecture that can be adopted by traffic modelers in the future for Oxford city. The limitations on this study are bountiful (lack of data, lack of computational resources, etc.) however it provides the grounds to further test this model and make future improvements. Using more traffic and weather data would most certainly improve the performance of this model and provide further insight into the model's strengths and weaknesses. Oxford City Council do collect quality traffic data, and this could be utilized in this model in future work. Also, in-situ data from city sensors or even autonomous vehicles could greatly enhance the predictive capabilities of this model and significantly improve the research. Ultimately the better the data quality and quantity, the better the prediction.

Regarding the model architecture, more layers and robust hyperparameter tuning also can be explored with the likely result being better performance. To achieve this a GPU should be used as training the models will be much quicker and efficient. Should this be achieved in the future, the next step would be to expand the study to incorporate large regions of Oxford city, increasing the sources of data, and to eventually incorporate the entire city.

All the objectives of this dissertation were achieved, along with the aim of the research. This study developed the base for a predictive short-term traffic model for Oxford and outlined many improvements that can be made in future research. The model architecture can be adjusted to incorporate many varying data streams and is flexible enough to be scaled up. Continued research on this topic could eventually lead to its integration into autonomous vehicle decision making when considering route planning and contribute significantly to ITSs in the Oxford area.

8.3 Personal Remarks and Learnt Lessons

This project provided an excellent learning experience in project management since this research required a complete life cycle development of a product. This included the research, administrative, data collection, data modeling, and analysis phases to successfully develop a traffic model for Oxford. Negative experiences also led to learnt lessons, including the administrative difficulties experienced during the data collection phase. Additionally, this project provided an excellent medium to improve programming skills and utilize new Python libraries. The most important takeaway from this dissertation however is the newly developed understanding of deep neural networks and complex neural network architectures, which is a key area of development in traffic prediction and data science in general. The dissertation was successful in that it achieved its aim and set up the base for further research into this project, should this occur in the future. It was a pleasure to study this topic and develop a traffic model, and the hope is that this research can be continued sometime in the near future.

9. References

Abadi, M. *et al.* (2016) 'Tensorflow: A system for large-scale machine learning',

Bruna, J. *et al.* (2013) 'Spectral Networks and Locally Connected Networks on Graphs'.

Castanedo, F. (2013) 'A Review of Data Fusion Techniques', *The Scientific World Journal,* 2013, pp. 704504. doi: 10.1155/2013/704504.

Chollet, F. and others (2015) 'keras',

Council, O. C. *Transport monitoring.* https://www.oxfordshire.gov.uk/residents/roads-and-transport/traffic/transport-monitoring?utm_term=nil&utm_content=: Oxfordshire City Council (Accessed: September 2).

Author (2018) *StellarGraph Machine Learning Library*.

David, E. R. and James, L. M. (1987) 'Learning Internal Representations by Error Propagation', *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*: MIT Press, pp. 318-362.

David, K. (2020) 'Oxfordshire's new transport model to offer 'more nimble' insights for network planners', pp. 1

DeLancey, J. (2018) 'Predictive Routing Using Traffic Patterns',

Essien, A. *et al.* (2021) 'A deep-learning model for urban traffic flow prediction with traffic events mined from twitter', *World Wide Web,* 24. doi: 10.1007/s11280-020-00800-3.

Fionda, V. and Palopoli, L. (2011) 'Biological Network Querying Techniques: Analysis and Comparison', *Journal of computational biology : a journal of computational molecular cell biology,* 18, pp. 595-625. doi: 10.1089/cmb.2009.0144.

Gardner, M. W. and Dorling, S. R. (1998) 'Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences', *Atmospheric Environment,* 32(14), pp. 2627-2636. doi: https://doi.org/10.1016/S1352-2310(97)00447-0.

Guo, S. *et al.* 'Attention Based Spatial-Temporal Graph Convolutional Networks for Traffic Flow Forecasting'. *AAAI*.

Günther, F. and Fritsch, S. (2010) 'neuralnet: Training of Neural Networks', *R Journal,* 2. doi: 10.32614/RJ-2010-006.

Author (2008) *Exploring network structure, dynamics, and function using NetworkX*.

Hashemi Fath, A., Madanifar, F. and Abbasi, M. (2020) 'Implementation of multilayer perceptron (MLP) and radial basis function (RBF) neural networks to predict solution gas-oil ratio of crude oil systems', *Petroleum,* 6(1), pp. 80-91. doi: https://doi.org/10.1016/j.petlm.2018.12.002.

Hochreiter, S. and Schmidhuber, J. (1997) 'Long Short-Term Memory', *Neural Computation,* 9(8), pp. 1735-1780. doi: 10.1162/neco.1997.9.8.1735.

Hunter, J. D. (2007) 'Matplotlib: A 2D graphics environment', *Computing in Science \& Engineering*, 9(3), pp. 90--95

IBM and Education, C. (2021) 'Overfitting',

Ioffe, S. and Szegedy, C. (2015) 'Batch normalization: accelerating deep network training by reducing internal covariate shift', *Proceedings of the 32nd International Conference on*

International Conference on Machine Learning - Volume 37. Lille, France: JMLR.org, pp. 448–456.
Jain, A. *et al.* (2015) 'Structural-RNN: Deep Learning on Spatio-Temporal Graphs'.
Jia, T. and Yan, P. (2020) 'Predicting Citywide Road Traffic Flow Using Deep Spatiotemporal Neural Networks', *IEEE Transactions on Intelligent Transportation Systems,* PP, pp. 1-11. doi: 10.1109/TITS.2020.2979634.
Jordan, J. (2018) 'Setting the learning rate of your neural network.',
Kipf, T. N. and Welling, M. (2017) 'Semi-Supervised Classification with Graph Convolutional Networks', pp.
Laranjeira, J. (2020) 'What is traffic prediction and how does it work?',
Lars, B. *et al.* (2011) 'Scikit - learn : Machine Learning in P ython', *Journal of Machine Learning Research,* 12, pp. 2825 -- 2830.
Lau, J. (2020) 'Google Maps 101: How AI helps predict traffic and determine routes',
Lee, K. *et al.* (2021) 'Short-Term Traffic Prediction With Deep Neural Networks: A Survey', *IEEE Access,* 9, pp. 54739-54756.
Author (2021) *logCurrentTimeline.gs* (Version 1).
Michael, W. and Olga, B. (2017) 'seaborn: v0.8.1',
Miglani, A. and Kumar, N. (2019) 'Deep learning models for traffic flow prediction in autonomous vehicles: A review, solutions, and challenges', *Vehicular Communications,* 20, pp. 100184. doi: https://doi.org/10.1016/j.vehcom.2019.100184.
Navmart (2021) *Conquer the roads with HERE Real-Time Traffic.* Navmart: Navmart (Accessed: September 2).
Niklas, D. (2021) 'A Guide to RNN: Understanding Recurrent Neural Networks and LSTM Networks',
Park, J., Yi, D. and Ji, S. (2020) 'A Novel Learning Rate Schedule in Optimization for Neural Networks and It's Convergence', *Symmetry,* 12(4). doi: 10.3390/sym12040660.
Paszke and Adam and Gross, S. a. M., Francisco and Lerer, Adam and Bradbury, James and Chanan, Gregory and Killeen, Trevor and Lin, Zeming and Gimelshein, Natalia and Antiga, Luca and Desmaison, Alban and Kopf, Andreas and Yang, Edward and DeVito, Zachary and Raison, Martin and Tejani, Alykhan and Chilamkurthy, Sasank and Steiner, Benoit and Fang, Lu and Bai, Junjie and Chintala, Soumith (2019) 'PyTorch: An Imperative Style, High-Performance Deep Learning Library',
Patro, S. G. and Sahu, D.-K. K. (2015) 'Normalization: A Preprocessing Stage', *IARJSET*. doi: 10.17148/IARJSET.2015.2305.
Raschka, S. (2015) 'Single-Layer Neural Networks and Gradient Descent',
Rosenblatt, F. 1957. The Perceptron: A Perceiving and Recognizing Automation. Cornell, New York: Cornell.
Salehinejad, H. *et al.* (2017) 'Recent Advances in Recurrent Neural Networks'.
Samit, B. 2019. Impact of Data Normalization on Deep Neural Network for Time Series Forecasting. *In:* Abhishek, D. (ed.).
Soua, R., Koesdwiady, A. and Karray, F. 'Big-data-generated traffic flow prediction using deep learning and dempster-shafer theory'. *2016 International Joint Conference on Neural Networks (IJCNN)*, 24-29 July 2016, 3195-3202.

Srivastava, N. *et al.* (2014) 'Dropout: a simple way to prevent neural networks from overfitting', *J. Mach. Learn. Res.,* 15(1), pp. 1929–1958.

Staudemeyer, R. and Morris, E. (2019) *Understanding LSTM -- a tutorial into Long Short-Term Memory Recurrent Neural Networks.*

Suh, B., Shao, Y. and Sun, Z. 'Vehicle Speed Prediction for Connected and Autonomous Vehicles Using Communication and Perception'. *2020 American Control Conference (ACC)*, 1-3 July 2020, 448-453.

Sun, S., Wu, H. and Xiang, L. (2020) 'City-Wide Traffic Flow Forecasting Using a Deep Convolutional Neural Network', *Sensors,* 20, pp. 421. doi: 10.3390/s20020421.

'Tableau (version. 9.1)', (2016) *Journal of the Medical Library Association : JMLA,* 104(2), pp. 182-183. doi: 10.3163/1536-5050.104.2.022.

Tedjopurnomo, D. A. *et al.* (2020) 'A Survey on Modern Deep Neural Network for Traffic Prediction: Trends, Methods and Challenges', *IEEE Transactions on Knowledge and Data Engineering*, pp. 1-1. doi: 10.1109/TKDE.2020.3001195.

Thakur, D. (2018) 'LSTM and its equations',

Tomorrow.io (2021) *Weather API Powered by Proprietary Technology.* Boston, MA: Tomorrow.io (Accessed: June 2, 2021).

Ujjwal, K. (2016) 'A Quick Introduction to Neural Networks',

Verma, K. and Singh, P. (2015) 'An Insight to Soft Computing based Defect Prediction Techniques in Software', *International Journal of Modern Education and Computer Science,* 7, pp. 52-58. doi: 10.5815/ijmecs.2015.09.07.

Warner, B. and Misra, M. (1970) 'Understanding Neural Networks as Statistical Tools', *The American Statistician,* 50. doi: 10.1080/00031305.1996.10473554.

Williams, B. M. and Hoel, L. (2003) 'Modeling and Forecasting Vehicular Traffic Flow as a Seasonal ARIMA Process: Theoretical Basis and Empirical Results', *Journal of Transportation Engineering-asce,* 129, pp. 664-672.

Wu, Z. *et al.* (2021) 'A Comprehensive Survey on Graph Neural Networks', *IEEE Transactions on Neural Networks and Learning Systems,* 32(1), pp. 4-24. doi: 10.1109/TNNLS.2020.2978386.

Yin, X. 2021. Deep Learning on Traffic Prediction: Methods, Analysis and Future Directions. *In:* Wu, G. *et al.* (eds.) v4 ed.

Yu, B., Yin, H. and Zhu, Z. (2018) *Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting.*

Yuan, F. *et al.* (2021) *Spatio-Temporal Graph Convolutional Networks for Road Network Inundation Status Prediction during Urban Flooding.*

Yuankai, W. and Tan, H. (2016) 'Short-term traffic flow forecasting with spatial-temporal correlation in a hybrid deep learning framework'.

Zhang, D. and Kabuka, M. (2018) 'Combining Weather Condition Data to Predict Traffic Flow: A GRU Based Deep Learning Approach', *IET Intelligent Transport Systems,* 12. doi: 10.1049/iet-its.2017.0313.

Zhao, L. *et al.* (2019) 'T-GCN: A Temporal Graph Convolutional Network for Traffic Prediction', *IEEE Transactions on Intelligent Transportation Systems,* PP, pp. 1-11. doi: 10.1109/TITS.2019.2935152.

10. Table Appendix

Hyperparameter Tuning Results												
Model	**LSTM**				**T-GCN**				**T-GCN-wx**			
Prediction Horizon	***5 min***	***15 min***	***30 min***	***60 min***	***5 min***	***15 min***	***30 min***	***60 min***	***5 min***	***15 min***	***30 min***	***60 min***
Learning Rate	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Data Fusion Layer Activation	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	linear	linear	linear	linear
GCN Activation	n/a	n/a	n/a	n/a	linear	linear	linear	linear	linear	linear	linear	linear
GCN Layer Neurons	n/a	n/a	n/a	n/a	10	15	15	15	10	15	15	15
LSTM Activation	linear	linear	linear	relu	linear	linear	relu	relu	linear	linear	relu	relu
LSTM Layer Neurons	200	200	200	100	200	200	100	100	200	200	100	100

Table 1: The hyperparameter tuning results from the LSTM, T-GCN, and T-GCN-wx models.

5-minute Traffic Prediction Performance			
	MSE	*MAE*	*R2*
Linear Regression	0.00283	**0.03323**	0.9147
LSTM	0.00309	0.03651	0.9085
T-GCN	**0.0028**	0.03572	**0.9167**
T-GCN-wx	0.0029	0.03518	0.9123

Table 2: 5-minute traffic prediction model results.

15-minute Traffic Prediction Performance			
	MSE	*MAE*	*R2*
Linear Regression	0.004630	**0.046004**	0.860439
LSTM	0.004654	0.046018	0.858499
T-GCN	0.004590	0.047303	0.861263
T-GCN-wx	**0.004564**	0.047146	**0.864136**

Table 3: 15-minute traffic prediction model results.

30-minute Traffic Prediction Performance			
	MSE	*MAE*	*R2*
Linear Regression	0.005088	0.048934	0.845005
LSTM	0.005256	0.050191	0.845525
T-GCN	**0.004734**	0.047694	**0.857816**
T-GCN-wx	0.004830	**0.047225**	0.855002

Table 4: 30-minute traffic prediction model results.

60-minute Traffic Prediction Performance			
	MSE	*MAE*	*R2*
Linear Regression	0.005689	0.051442	0.823471
LSTM	0.005429	0.050152	0.831871
T-GCN	0.005299	0.050152	0.838926
T-GCN-wx	**0.005288**	**0.049628**	**0.841170**

Table 5: 60-minute traffic prediction model results.

11. Code Appendix

All code available in the GitHub repository:

https://github.com/ldicarlo1/development_of_traffic_model_for_Oxford

[1]

HERE_Traffic_API.py

```
import numpy as np
import requests
from bs4 import BeautifulSoup
from xml.etree.ElementTree import XML, fromstring, tostring
from datetime import datetime
import pandas as pd
import sys
import matplotlib.pyplot as plt


class HERE_Traffic():
    def __init__(self):
        # collect current datetime
        now = datetime.now()
        self.timestamp = now.strftime("%Y%m%d%H%M")

    def credentials(self, APP_ID, APP_CODE):
        '''Please enter HERE API credentials:

        API_ID : Enter HERE account application identification.
        APP_CODE : Enter HERE account application code.
        API_KEY: Enter ONLY for traffic incident data. Default value is null.
        '''
        self.APP_ID = APP_ID
        self.APP_CODE = APP_CODE

    def bbox(self, lat0, lon0, lat1, lon1):
        '''Bounding-box information for traffic flow.
        '''
        self.lat0 = lat0
        self.lon0 = lon0
        self.lat1 = lat1
        self.lon1 = lon1
```

```
    def _connect_traffic_flow(self):
        '''Connects to HERE Traffic API using bbox and API credentials. Returns parsed
XML response.
        If invalid credentials are used returns error message.
        '''

        page =
requests.get(f'https://traffic.api.here.com/traffic/6.3/flow.xml?app_id={self.APP_ID}&
app_code={self.APP_CODE}&bbox={self.lat0},{self.lon0};{self.lat1},{self.lon1}&response
attributes=sh,fc')

        # if credentials are incorrect, prompt error message and kill program.
        if str(page) == '<Response [401]>':
            sys.exit("Invalid credentials. Please re-enter correct API identification
number or code.")
        else:
            soup = BeautifulSoup(page.text, "lxml")
            response = soup.find_all('fi')

        return response

    def _connect_incident_reports(self):
        '''Connects to HERE Incident API using bbox and API credentials. Returns
parsed html response.
        '''
        page =
requests.get(f'https://traffic.api.here.com/traffic/6.3/incidents.xml?app_id={self.APP
_ID}&app_code={self.APP_CODE}&bbox={self.lat0},{self.lon0};{self.lat1},{self.lon1}&res
ponseattributes=sh,fc')
        soup = BeautifulSoup(page.text, "html.parser")
        parsed = soup.find_all("trafficml_incidents")[0]

        return parsed


    def traffic_flow(self):
        '''
        Returns traffic flow information, latitudes, longitudes, road names.
        '''
        # call traffic response
        response = self._connect_traffic_flow()
        # loop through each road and collect road type, road speed limit, actual real-
time
        # road speed, road name, and traffic direction.
        a1=[]
        loc_list_hv=[]
        lats=[]
```

```
        lons=[]
        speed_uncapped=[]
        speed_capped=[]
        jam_factor=[]
        free_flow_spd=[]
        names = []
        direction=[]
        c=0
        for html_response in response:
            #for j in range(0,len(shps)):
            xml_response = fromstring(str(html_response))
            fc=5
            for road in xml_response:
                if('fc' in road.attrib):
                    fc=int(road.attrib['fc'])
                if('cn' in road.attrib):
                    cn=float(road.attrib['cn'])
                if('su' in road.attrib):
                    su=float(road.attrib['su'])
                if('sp' in road.attrib):
                    sp=float(road.attrib['sp'])
                if('jf' in road.attrib):
                    jf=float(road.attrib['jf'])
                if('ff' in road.attrib):
                    ff=float(road.attrib['ff'])
                if('de' in road.attrib):
                    de=(road.attrib['de'])
                if('qd' in road.attrib):
                    qd=(road.attrib['qd'])

            # split road information into individual latitude/longitude arrays based
on road shape
            # fc is highways and major roadways. CN is confidence in real-time traffic
flow information (max 1). At least 70%.
            if((fc<=5) and (cn>=0.7)):

                # road shapes by lat/lon coordinates
                shps=html_response.find_all("shp")

                for j in range(0,len(shps)):
                    latlong=shps[j].text.replace(',',' ').split()
                    #loc_list=[]
                    la=[]
                    lo=[]
                    su1=[]
                    ff1=[]
                    sp1=[]
```

```
                    jf1=[]
                    qd1 = []
                    name=[]

                    for i in range(0,int(len(latlong)/2)):
                        # organized as pairs (lat , lon) therefore split by lat/lon
values.

loc_list_hv.append([float(latlong[2*i]),float(latlong[2*i+1]),float(su),float(ff)])
                        la.append(float(latlong[2*i]))
                        lo.append(float(latlong[2*i+1]))
                        su1.append(float(su))
                        ff1.append(float(ff))
                        sp1.append(float(sp))
                        jf1.append(float(jf))
                        name.append(de)
                        qd1.append(qd)
                    lats.append(la)
                    lons.append(lo)
                    speed_uncapped.append(np.mean(su1))
                    speed_capped.append(np.mean(sp1))
                    jam_factor.append(np.mean(jf1))
                    free_flow_spd.append(np.mean(ff1))
                    names.append(str(de))
                    direction.append(qd1[0])

        return names, direction, lats, lons, speed_capped ,speed_uncapped,
free_flow_spd, jam_factor

    def incident_report(self):
        ''' Returns incident type (construction, roadwork, accident, etc) along with
coordinates of the incident location and the roads affected.
        '''
        # call incident response
        response = self._connect_incident_reports()

        # collect the incident description, status, coordinates, and street path
coordinates
        descs = response.find_all("traffic_item_type_desc")
        stat = response.find_all("traffic_item_status_short_desc")
        point_lat = response.find_all("latitude")
        point_lon = response.find_all("longitude")
        shps = response.find_all("shapes")

        # predefine variables
        lats=[]
        lons=[]
```

```
        latlong = []
        descriptions = []
        status = []
        latitude = []
        longitude = []

        # loop and collect the incident information for each case
        for j in range(0,len(shps)):
            latlong=shps[j].text.replace(',',' ').split()
            desc = descs[j].text
            stats = stat[j].text
            point_lats = point_lat[j].text
            point_lons = point_lon[j].text
            la=[]
            lo=[]
            # remove combined latlon values that are duplicates and incorrectly listed
            [latlong.remove(x) for x in latlong if len(x)>10]

            # assign each lat and lon point
            for i in range(0,int(len(latlong)/2)):
                la.append(float(latlong[2*i]))
                lo.append(float(latlong[2*i+1]))

            # append the information into lists
            lats.append(la)
            lons.append(lo)
            descriptions.append(desc)
            status.append(stats)
            latitude.append(point_lats)
            longitude.append(point_lons)

        return descriptions, status, latitude, longitude, lats, lons

    def generate_traffic_csv(self, outdir):
        '''This function generates a CSV containing traffic variables for each road
segment in the specified bbox.
        Please specify an out directory.
        '''
        # call variables from traffic flow function
        names, qd, lats, lons, speed_capped ,speed_uncapped, free_flow_spd, jam_factor
= self.traffic_flow()

        # store variables in dataframe
        traffic_df = pd.DataFrame({"road_name": names, "direction":qd,"lats":lats,
"lons":lons, "sp":speed_capped, "su":speed_uncapped,"ffs":free_flow_spd,
"jf":jam_factor})
        traffic_df.sp = traffic_df.sp.round(0)
```

```
        traffic_df.su = traffic_df.su.round(0)
        traffic_df.ffs = traffic_df.ffs.round(0)

        # create CSV file containing the variables
        traffic_df.to_csv(f"{outdir}/oxford_traffic_{self.timestamp}.csv")

    def generate_incident_csv(self, outdir):
        '''This function generates a CSV containing incident variables for each road
segment in the specified bbox.
        Please specify an out directory.
        '''
        # call variables from incident report function
        descriptions, status, latitude, longitude, lats0, lons0 =
self.incident_report()

        # store variables in dataframe
        incident_df = pd.DataFrame({"status":status, "desc":descriptions,
"point_lat":latitude, "point_lon":longitude,
        "lats":lats0, "lons":lons0})

        # create CSV file containing the variables
        incident_df.to_csv(f"{outdir}/oxford_incident_{self.timestamp}.csv")
```

[2]

data_preprocessing.py

```
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
import matplotlib
matplotlib.use('TKAgg')
import matplotlib.pyplot as plt
from matplotlib.backends.backend_agg import FigureCanvas
import ast
from skimage import data, color


def _geographic_subset_data():
    '''This function returns a subset of the Oxford traffic data for Oxford by
indexing based on a lon/lat bbox.
    '''

    # loop thru and convert lat/lon strings to python lists
    loni = []
    lati=[]
    for i in range(len(traffic_matrix)):
        lati.append(ast.literal_eval(traffic_matrix.lats.iloc[i]))
```

```
        loni.append(ast.literal_eval(traffic_matrix.lons.iloc[i]))

    traffic_matrix['lons'] = loni
    traffic_matrix['lats'] = lati

    # index the data based on the lat/lon bounding coordinates provided in the
initialization of the function.
    bbox_data=[]
    for i in range(len(traffic_matrix)):
        lonmin = np.array(traffic_matrix.lons.iloc[i]).min()
        lonmax = np.array(traffic_matrix.lons.iloc[i]).max()
        latmin = np.array(traffic_matrix.lats.iloc[i]).min()
        latmax = np.array(traffic_matrix.lats.iloc[i]).max()
        if latmax<=top and latmin >=bottom and lonmin >= left and lonmax <=right:
            bbox_data.append(traffic_matrix.iloc[i].values)
    bbox_data = pd.DataFrame(np.array(bbox_data))
    bbox_data.columns = traffic_matrix.columns

    return bbox_data
```

[3]

load_traffic_data.py

```
from classes.HERE_Traffic_API import HERE_Traffic
import os

# get cwd
path = os.getcwd()

# HERE API credentials
APP_ID = "ZAJVrYvNkkrgRxb5HbBl"
APP_CODE = "KUe4Gt19T3YFNm3iw_QSZQ"

# latitude/longitude bounds of Oxford, UK
lat0 = 51.730406
lon0 = -1.292902
lat1 = 51.804694
lon1 = -1.202052
```

```
# call HERE traffic function and create CSV files containing traffic flow & incident
data
t = HERE_Traffic()
t.credentials(APP_ID,APP_CODE)
t.bbox(lat0,lon0,lat1,lon1)

# store the files in an outdirectory
t.generate_traffic_csv(f"{path}/data_collection/data/traffic_data")
t.generate_incident_csv(f"{path}/data_collection/data/incident_data")
```

[4]

preprocessing.py

```
import numpy as np

''' Lists of functions for preprocessing data before modeling.
'''

def train_test_split(data, train_portion):
    '''Function splits data into testing and training using a split ratio.

    Input Args:

    data (array-obj): traffic or weather data

    train portion (float): percentage of training data to be split (value between 0 or
1)

    Return Args:

    train data (array-obj): split of training data

    test data (array-obj): split of testing data

    '''
    time_len = data.shape[1]
    train_size = int(time_len * train_portion)
    train_data = np.array(data[:, :train_size])
    test_data = np.array(data[:, train_size:])

    return train_data, test_data

def scale_data(train_data, test_data):
    '''Function normalizes the data between 0 and 1.
```

```
    Input Args:

    train_data (array-obj): training traffic or weather data to be scaled (value
between 0 or 1)

    test_data (array-obj): testing traffic or weather data to be scaled (value between
0 or 1 )

    Return Args:

    train scaled (array-obj): scaled training data

    test scale (array-obj): scaled testing data

    '''

    train_max = train_data.max()
    train_min = train_data.min()

    train_scaled = (train_data - train_min) / (train_max - train_min)
    test_scaled = (test_data - train_min) / (train_max - train_min)

    return train_scaled, test_scaled

def sequence_data_preparation(seq_len, pre_len, train_data, test_data):
    '''Function prepares a sequence for the LSTM input by creating a sequence of
predictors and a target prediction N steps in advance.
    Input Args:

    seq_len (float): number of timesteps into the past to predict the future time
series value.

    pre_len (float): number of timesteps into the future that the model should aim to
predict.

    train_data (array-obj): training data

    test_data (array-obj): testing data


    Return Args:

    trainX (array-obj): training predictors
    trainY (array-obj): training predictions
    testX (array-obj): testing predictors
    testY (array-obj): training predictions
```

```
    '''

    trainX, trainY, testX, testY = [], [], [], []

    for i in range(train_data.shape[1] - int(seq_len + pre_len - 1)):
        a = train_data[:, i : i + seq_len + pre_len]
        trainX.append(a[:, :seq_len])
        trainY.append(a[:, -1])

    for i in range(test_data.shape[1] - int(seq_len + pre_len - 1)):
        b = test_data[:, i : i + seq_len + pre_len]
        testX.append(b[:, :seq_len])
        testY.append(b[:, -1])

    trainX = np.array(trainX)
    trainY = np.array(trainY)
    testX = np.array(testX)
    testY = np.array(testY)

    return trainX, trainY, testX, testY
```

[5]

Oxfordcity_wx_variables_collection.gs

```
function logCurrentTimeline() {
  // set the Timelines POST endpoint as the target URL
  var postTimelinesURL = "https://api.tomorrow.io/v4/timelines";

  // get your key from app.tomorrow.io/development/keys
  var apikey = "HM0APtY7mVlmZXSKdBGOwIf0lf61rBXp";

  // latitude and longitude of Oxford
  var lat = 51.754325444116134;
  var lon = -1.2577268845204246;

  // request the "current" timelines with all the query string parameters as options
  var postTimelinesParameters = {
    location: [lat, lon],
    fields: [
      "precipitationIntensity",
```

```
      "precipitationType",
      "windSpeed",
      "temperature",
      "humidity",
      "cloudCover",
      "weatherCode",
    ],
    units: "metric",
    timesteps: ["current"],
    timezone: "UTC",
  };

  var response = JSON.parse(UrlFetchApp.fetch(postTimelinesURL + `?apikey=${apikey}`,
{
    method: "post",
    contentType: 'application/json',
    payload: JSON.stringify(postTimelinesParameters),
  }).getContentText());

  // sort current interval values according to fields order
  var values = postTimelinesParameters.fields.map(field =>
response.data.timelines[0].intervals[0].values[field])
  var timestamp = response.data.timelines[0].intervals[0].startTime;

  // get active spreadsheet
  var sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();

  // delete and prepend fields row (added to this cron job and not as a prerequisite,
for guide simplicity)
  sheet.deleteRow(1)
  postTimelinesParameters.fields.unshift("timestamp");
  sheet.insertRowBefore(1).getRange(1, 1, 1,
postTimelinesParameters.fields.length).setValues([postTimelinesParameters.fields]);

  // append values of last Timelines run
  sheet.appendRow([timestamp, ...values]);
  // make sure the cell is updated right away in case the script is interrupted
  SpreadsheetApp.flush();
}
```

[6]

Modeling_5min_prediction.py

```
# change directory to parent folder to access all folders
import os
path = os.path.dirname(os.getcwd())
os.chdir(path)
from data_preprocessing.classes.load_traffic_data import Import_Traffic_Data

import networkx as nx
import pandas as pd
import numpy as np
import ast
import math
import keras
import matplotlib.pyplot as plt
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dropout, Dense,concatenate
from stellargraph import StellarGraph, StellarDiGraph
from stellargraph.layer import GCN_LSTM
import stellargraph as sg
from datetime import datetime
from keras.models import Sequential, load_model
from keras.layers import LSTM
from pmdarima.model_selection import train_test_split
import seaborn as sns
from classes import model_performance,preprocessing
import pickle
import time as timex


################################################
# Load traffic data
################################################
# Peartree roundabout bbox and datetimes of interest
top=51.798433
bottom=51.791451
right=-1.281979
left=-1.289524
datetime_start=datetime(2021,6,23,0,0)
datetime_end=datetime(2021,7,13,10,50)

# load in traffic data
traffic_data,time =
Import_Traffic_Data(top,bottom,right,left).load_traffic_data(datetime_start,datetime_e
nd)
```

```
################################################
# Load weather data
################################################
# load in 5min wx data from csv
wx_df =
pd.read_csv("data_collection/data/wx_data/oxfordcity_wx_variables_5min_intervals.csv")

# collect variables of significance
wx_vars = wx_df[['precipitationIntensity','temperature','humidity','weatherCode']]

wx_vars_scaled = np.zeros_like(wx_vars)

# normalize between 0 and 1
for i in range(4):
    norm = (wx_vars.iloc[:,i] - wx_vars.iloc[:,i].min())/(wx_vars.iloc[:,i].max() -
wx_vars.iloc[:,i].min())
    wx_vars_scaled[:,i] = norm
    #print(wx_vars_scaled[i].max())


# transpose data to be in proper format for preprocessing
wx_vars_scaled = wx_vars_scaled.T




################################################
# Create graph
################################################

# load in csv of node connections
connections = pd.read_csv(f"{path}/data_preprocessing/peartree_roundabout.csv")
connections.head(5)

# convert feeding roads to integers
for i in range(len(connections)):
#for i in range(4):
    try:
        connections.feeding_roads.iloc[i] =
ast.literal_eval(connections.feeding_roads.iloc[i])
    except ValueError:
        connections.feeding_roads.iloc[i] = np.nan

# node connections
nodes = connections["Unnamed: 0"]
```

```
roads = connections.feeding_roads

# replace nans with 0's
connections.feeding_roads = connections.feeding_roads.fillna(0)

# loop thru and establish edges
edge_list = []
for row in range(len(roads)):
    node1 = connections["Unnamed: 0"].iloc[row]
    node2 = connections.feeding_roads.iloc[row]
    try:
        for i in range(len(node2)):
            edge_list.append([node2[i], node1])
        #node2 = connections.feeding_roads.iloc[row]
    except TypeError:
        edge_list.append([node2, node1])

# remove 0's
edges = []
for edge in edge_list:
    if edge[0]==0:
        pass
    else:
        edges.append(edge)

#build the graph
G = nx.Graph()
for i in range(len(nodes)):
    G.add_node(nodes[i],spd=sp[:,i])
G.add_edges_from(edges)

# get adjacency matrix
A = nx.to_numpy_array(G)

# convert graph to stellargraph object for modeling
square = StellarGraph.from_networkx(G,node_features="spd")

# get feature matrix
X = square.node_features()

################################################
# Preprocess data
################################################
# specify the training rate
train_rate = 0.8

# replace missing values with nans
```

```
X = np.where(X<0,0,X)

# split train/test
train_data, test_data = preprocessing.train_test_split(X, train_rate)
wx_train_data, wx_test_data = preprocessing.train_test_split(wx_vars_scaled,
train_rate)

print("Train data: ", train_data.shape)
print("Test data: ", test_data.shape)

# scale data based on max/min
train_scaled, test_scaled = preprocessing.scale_data(train_data, test_data)

# create new train/test variables for wx variables
wx_train_data_ = []
wx_test_data_ = []

# loop thru and assign the wx data to each node
for i in range(70):
    wx_train_data_.append(wx_train_data.T)
    wx_test_data_.append(wx_test_data.T)

# convert data to correct shape
wx_train_data_ = np.array(wx_train_data_)
wx_test_data_ = np.array(wx_test_data_)

################################################
# 5-min prediction length modeling preprocessing
################################################
# the number of timesteps up to the prediction that we will feed to the model (5-
minute intervals)
seq_len = 12

# the amount of time in advance we want to predict (5-minute intervals)
pre_len = 1

# preprocess traffic data
traffic_trainX, trainY, traffic_testX, testY =
preprocessing.sequence_data_preparation(
    seq_len, pre_len, train_scaled, test_scaled
)

# preprocessing weather data
wx_trainX, wx_trainY, wx_testX, wx_testY = preprocessing.sequence_data_preparation(
    seq_len, pre_len, wx_train_data_, wx_test_data_
)
```

```
#Combine the weather variables w/ traffic matrix to create feature matrix
trainX =
np.empty((len(wx_trainX[:,0,0,0]),len(wx_trainX[0,:,0,0]),len(wx_trainX[0,0,:,0]),5) )
testX =
np.empty((len(wx_testX[:,0,0,0]),len(wx_testX[0,:,0,0]),len(wx_testX[0,0,:,0]),5) )

trainX[:,:,:,0] = traffic_trainX
trainX[:,:,:,1:5] = wx_trainX
testX[:,:,:,0] = traffic_testX
testX[:,:,:,1:5] = wx_testX

################################################
# Generate models
################################################


############## Linear Regression ##################
# define linear regression model
lr = LinearRegression()

# get the number of road segments
num_road_seg = trainX[0,:,0]

# make empty list to populate with predictions from each road segment
predictions = []

# loop thru each road segment and train the ML algorithm
for road_seg in range(len(num_road_seg)):

    # model the ML algorithm with the training/testing data
    lr.fit(trainX[:,road_seg,:], trainY[:,road_seg])

# save the model to disk
filename = 'modeling/models/lr-5min.sav'
pickle.dump(lr, open(filename, 'wb'))


############## LSTM ##################
# define model
model_lstm = Sequential()
model_lstm.add(LSTM(200, activation='linear', input_shape=(70, 12,)))
model_lstm.add(Dense(70))

# compile model
optimizer = keras.optimizers.Adam(lr=0.001)
model_lstm.compile(optimizer=optimizer, loss="mse", metrics=["mse"])
```

```
# print summary
model_lstm.summary()
sg.utils.plot_history(history)

# save model to folder
model_lstm.save('modeling/models/lstm-5min')

############## T-GCN ##################
gcn_lstm = GCN_LSTM(
    seq_len=seq_len,
    adj=A,
    gc_layer_sizes=[10],
    gc_activations=["linear"],
    lstm_layer_sizes=[200],
    lstm_activations=["linear"],
    dropout=0.0,
)
# model architecture with keras
x_input, x_output = gcn_lstm.in_out_tensors()
model_tgcn = Model(inputs=x_input, outputs=x_output)

# compile model
optimizer = keras.optimizers.Adam(lr=0.001)
model_tgcn.compile(optimizer=optimizer, loss="mse", metrics=["mse"])

history = model_tgcn.fit(
    x=trainX,
    y=trainY,
    epochs=75,
    batch_size=64,
    shuffle=True,
    verbose=1,
    validation_data=(testX,testY)
)
# plot loss
sg.utils.plot_history(history)

# save model to folder
model_tgcn.save('modeling/models/tgcn-5min')

############## T-GCN-wx ##################
gcn_lstm = GCN_LSTM(
    seq_len=seq_len,
    adj=A,
    gc_layer_sizes=[10],
    gc_activations=["linear"],
    lstm_layer_sizes=[200],
```

```
    lstm_activations=["linear"],
    dropout=0.1
)
# build data fusion layer which will merge wx/traffic feature matrix (length 5) into
one array to be fed into the t-gcn model
input_layer = Input(shape=(70,12,5))
layer1 = Dense(1, activation='linear')(input_layer)
layer2 = BatchNormalization()(layer1)
layer3 = Dropout(0.1)(layer1)
output_layer = Reshape((70,12))(layer3)
data_fusion_model = Model(input_layer,output_layer)

# recall tgcn model
x_input, x_output = gcn_lstm.in_out_tensors()
tgcn_model = Model(inputs=x_input, outputs=x_output)


# of data fusion model feeds into t-gcn
output = tgcn_model(data_fusion_model.output)

# define entire model
tgcn_wx_model = Model(data_fusion_model.input,output, name="T-GCN-WX")
tgcn_wx_model.summary()

# compile model
optimizer = keras.optimizers.Adam(lr=0.001)
tgcn_wx_model.compile(optimizer=optimizer, loss="mse", metrics=["mse"])

history = tgcn_wx_model.fit(
    x=trainX,
    y=trainY,
    epochs=100,
    batch_size=64,
    shuffle=True,
    verbose=1,
    validation_data=(testX,testY)
)

sg.utils.plot_history(history)

# save model to folder
tgcn_wx_model.save('modeling/models/tgcn_wx-5min')
```