Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Traffic Forecasting in Dublin City using Graph Neural Networks:

Investigating the use of an STGCN Model in Urban Traffic Flow Prediction

by
Oscar Crowley

A dissertation submitted to the University of Dublin in partial fulfilment of the requirements for the Degree of

Magister in Arte Ingeniaria

Trinity College Dublin

April 14, 2022

Declaration of authorship

I declare that this thesis has not been submitted as an exercise for a degree at this or any other university, and it is entirely my own work.

I agree to deposit this thesis in the University's open access institutional repository or allow the Library to do so on my behalf, subject to Irish Copyright Legislation and Trinity College Library conditions of use and acknowledgement.

I consent to the examiner retaining a copy of the thesis beyond the examining period, should they so wish (EU GDPR May 2018).

Signature:

Date: April 14th, 2022

Acknowledgments

Firstly, I would like to thank my family for supporting me throughout my college career and feigning interest in my field of study. I would be nowhere without you. You don't have to read the rest, I appreciate that you made it this far.

Thank you to my flatmates for keeping me relaxed when I was panicking about this project and keeping me panicked when I was too relaxed.

To the residents of 55 Glenmalure Square, the jury is still out regarding your net impact on this thesis, but thanks anyway.

I am very grateful to my supervisor, Prof. Bidisha Ghosh, for giving clear guidance while allowing me to explore the avenues that interested me. The field of Graph Neural Networks has huge potential in many fields, and I am glad I was able to use this project to learn so much about them.

Special thanks to fellow students John Scanlon, Harry Humes, and Tom Boileau, for their help and advice on several aspects of this thesis.

Lastly I will thank the Schools of Engineering and Computer Science here at Trinity, along with all the lecturers and tutors that have helped me along the way.

Abstract

With the implementation of more sophisticated 'smart-city' technologies, traffic data has become more abundant and also more actionable. Despite this, traffic is still a huge problem in most cities, wasting both time and resources, while also damaging our environment. Understanding the spatio-temporal patterns of traffic can help us to forecast more accurately and adjust our infrastructure accordingly. Deep learning has seen huge success in this field of late, due to its ability to find patterns across space and time in non-obvious ways. More recently, Graph Neural Networks (GNNs) have been shown to improve on the state of the art, as they are able to model the innate graph structure of traffic networks. In this project, I apply a Spatio-Temporal Graph Convolutional Network (STGCN) model on Dublin City data in order to predict traffic flow, using flow and density as our inputs. To the best of my knowledge, this is the first time that a Graph Neural Network has been used for urban traffic forecasting in Dublin. I train STGCN models on 3 subsets of traffic junctions in the city centre and compare the performance against a Support Vector Regressor (SVR) model. These STGCN models outperform our baseline in most cases. I also create and evaluate a version of the model that outputs an uncertainty distribution around our estimate. This model was able to accurately represent the uncertainty of its estimates, even on unseen data. Lastly, I create a model that uses flow alone as its input. This one channel model does not perform as well as our two channel model, demonstrating that density plays a role in predicting flow in our experiments.

Contents

List of Figures

List of Tables

List of abbreviations

ARIMA	Auto-Regressive Integrated Moving Average
CNN	Convolutional Neural Network
DL	Deep Learning
ES	Einstein Summation
FCD	Floating Car Data
GCN	Graph Convolutional Network
ITS	Intelligent Transport System
LSTM	Long Short Term Memory
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
NLP	Negative Log Probability
RCD	Roadside Car Data
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
SCATS	Sydney Coordinated Adaptive Traffic System
STGCN	Spatio-Temporal Graph Convolutional Network
STGNN	Spatio-Temporal Graph Neural Network
SVR	Support Vector Regressor

Chapter 1

Introduction

1.1 Background

Traffic congestion is a huge problem faced by ever-expanding modern cities. It leads to air and noise pollution, increased cost of travel and freight, and increased numbers of car crashes, along with millions of human-hours lost every year. In the United States alone, traffic congestion was estimated to cost the economy $305 billion (22). This figure is calculated from the lost productivity of workers, increased cost of shipping goods, and wasted fuel. It does not even take into account the environmental costs associated. Despite its relatively small size, Dublin City is heavily affected. Dublin is the 16th most congested city in the world, with the average driver losing 89 hours to traffic in 2021 (21).

There are three primary methods of reducing traffic congestion; improving physical infrastructure, changing methods of transportation, and managing traffic flow in real time (25). The first two are limited by many socio-economic factors, while the third is more easily impacted by improved understanding and technology. This understanding has improved hugely over recent decades, thanks to increased data and new forecasting methods. With reliable forecasting, we could use Intelligent Transport Systems (ITSs) that would redirect traffic and manage traffic lights appropriately. A better understanding of the spatio-temporal patterns of traffic would allow us to divert flow for increased periods of traffic or more effectively tailor our infrastructure. Sydney Coordinated Adaptive Traffic System (SCATS) is the soft-

ware used by Dublin City Council to monitor traffic lights around the city. It logs several metrics of traffic passing over inductive loop sensors.

Graph Neural Networks (GNNs) are a type of Deep Learning model that perform operations on graph embeddings in order to model interconnected data points. These have a wide range of uses and have seen recent success in the field of traffic forecasting. By structuring the road network as a graph we can depict how connected two data points (for example, vehicle sensor) are by the weight of the edge between them. This allows us to capture how spatial traffic patterns change over time.

1.2 Objective of Work

The primary objective of this project is to create, train, and evaluate a GNN model for predicting traffic flow using our SCATS. To the best of my knowledge, this is the first GNN model created to forecast traffic in Dublin City. I will also need to implement a baseline model against which to compare my GNN. Most papers on GNN models for traffic forecasting operate on standard highway datasets, which gives little insight into how the models would perform in an urban setting.

My secondary objective is to thoroughly explain the operation of the GNN I implement. Descriptions of GNNs in academic papers are often vague, giving an overview of the types of layers used, but giving no context as to things like how this function is initialized or what variables are inputted or outputted from each layer. This would allow someone to expand upon the GNN I create, without having to interpret and incorporate information from several sources on how each component operates, or debug the code to find what the size of a particular array should be.

My tertiary objective is to investigate the data being used. This comes as a result of the evaluation of my model. I aim to use the evaluation to gain insights about the junctions and testing periods being used.

1.3 Dissertation Structure

1.3.1 Literature Review

In this section I will look briefly at different methods that have been used in traffic forecasting, including Deep Learning methods. I then give an overview of the history and theory of GNNs generally. Finally, I compare GNN structures that have been used in traffic forecasting. I will decide upon my own model structure at this point.

1.3.2 Methodology

The first two parts of the methodology describe how I collect and preprocess the data used by my GNN. Next, I give a thorough description of how the GNN operates. This includes a description of each process in the structure, along with the parameters used. I then describe my baseline model and training details.

1.3.3 Experiments & Results

I describe the time periods used for my train, validation, and testing data. Next, I describe the 3 sets of sensors that I will be evaluating my model performance on. The results of these evaluations are then displayed and compared to my baseline model. I then perform experiments using two substantively different models; one to investigate uncertainty in my predictions, and one to investigate the effect of flow and density in my predictions. I then discuss the results found in each of these cases.

1.3.4 Conclusion

The conclusion summarizes my findings and suggests areas for future research.

1.4 Ethical Considerations

In terms of data collection, this project is quite safe ethically. The loop detectors used by Dublin City Council and the SCATS system do not receive any information about the vehicles that are being measured, other than their presence. This is not the case with Floating Car Data (FCD) gathered from vehicles or mobile phones. In those situations, it might be possible to ascertain a user's identity from their travel

data even if the data were anonymized. In the case of traffic cameras, it might be possible to track movement patterns of an individual through facial or numberplate recognition.

One consideration we should be aware of is how these models could be used in the future. It is possible traffic flow and human flow forecasting could be used by governments to control crowds and protests. Another possibility is that traffic patterns could be maliciously used to justify the restructuring of city infrastructure. This could be done in order to rezone certain areas to the detriment of residents.

Chapter 2

Literature Review

In my Literature Review I will first discuss how traffic flow has traditionally been modelled. Next, I will describe the Deep Learning methods that have been used to model both the spatial and temporal patterns of traffic flow. I will give an overview on the relatively new field of Graph Neural Networks (GNNs) and specifically on how graph convolutions work. Finally, I will compare GNN models that have been used in traffic forecasting, and explain the choice of model for my application on Dublin City.

2.1 Traffic Theory

There is extensive literature on the theoretical models of traffic. Traditionally traffic is thought of like a fluid, with flow, density, and speed, though this is more true in highway setting with comparatively less noise than cities. Data in urban areas is obscured by many sources of noise making it inherently more difficult to predict than in highway settings.

The main metrics in traffic prediction are:

1. Flow (Q): The number of vehicles that passes through a given point in a given period of time. This is a more regular signal than the others in general.

2. Speed (S): The average speed of vehicles passing through a given point.

3. Density (K): The degree to which vehicles occupy the road space.

4. Demand: Number of users of a service who will request it in the area during a given period.

5. Travel Time: The amount of time it takes for someone to travel between two points. With the increase of floating vehicle data, travel time is also becoming a prevalent metric.

Flow, speed and density are linked through the Macroscopic Fundamental Diagram (MFD) shown in Figure 2.1. As the density increases the flow also increases since, if we don't have to slow down, more cars per metre means more cars per hour. However, after the critical density, increased density causes a decrease in flow. This is the congestion phase.

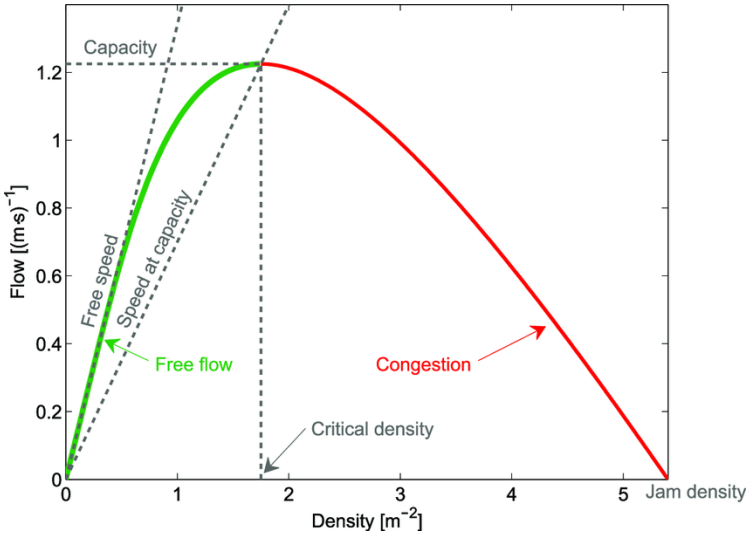


Figure 2.1: Macroscopic Fundamental Diagram of Traffic Flow (13)

Figure 2.2 shows a plot of the flow and density from one of our real junctions, demonstrating that the behaviour is not as clean in real urban settings. There is a low more variability within individual cars and times of day.

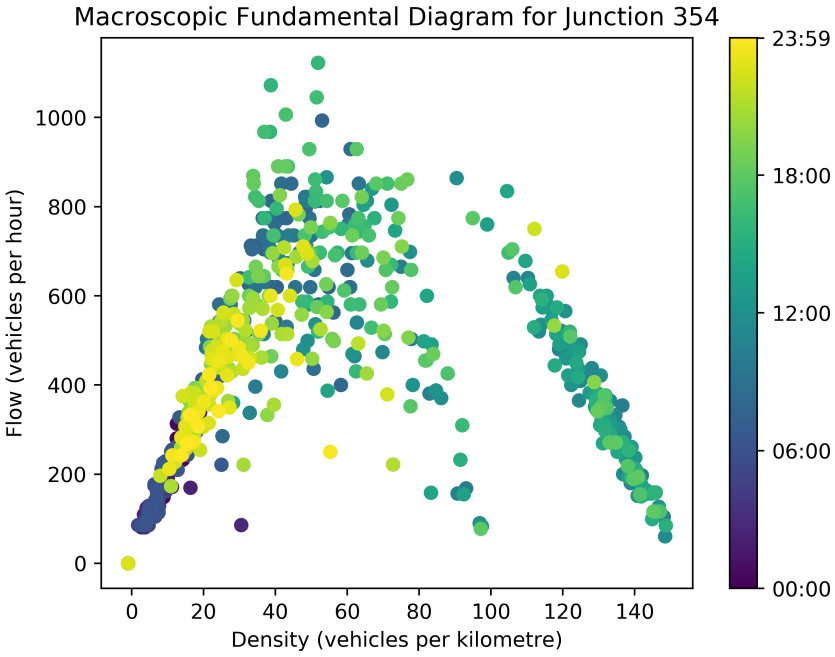


Figure 2.2: MFD on our SCATS data

2.2 Classical Methods

The earliest parametric approaches were developed in the 1970s and involved approaches such as time series analysis (30), or Kalman filters (31). Before these there were some largely unsuccessful naïve approaches such as historic average or instant values. Time series analysis involves using historical discrete-time data to predict future value. An example of this is the Auto-Regressive Integrated Moving Average (ARIMA) model, which uses a linear combination of errors from prior time steps to forecast. This can maintain variance while altering the mean, which can occur cyclically, such as in traffic patterns. Kalman filtering is a recurrent algorithm for state prediction given a large amount of uncertainty or noise in the measurement data.

Root Mean Square Error (RMSE), Mean Absolute Percentage Error (MAPE), and Mean Average Error (MAE) are used as the standards for assessing model performance (25). However, these can be misleading as, in network wide models, errors may propagate through time and space.

2.3 Machine Learning

Owing to the complexity and size of the traffic data being processed, machine learning has seen huge success in this field. Traffic data is largely non-stationary and non-linear (34) making it difficult to map with parametric statistical models. The early machine learning models used were Support Vector Regression (SVR) (5) and Random Forests (23). These involve creating a feature space and dividing it appropriately to make our predictions.

Many machine learning methods have been used since to approach this problem; fuzzy neural networks (35), fruit fly optimization (6), and K-Nearest Neighbours (36). These models showed some large improvements on the classical models, but traditional machine learning models suffer from asserting strong static assumptions on our input data and struggling to handle heavily non-linear temporal dependency (26).

2.4 Deep Learning in Traffic Forecasting

Deep learning involves the stacking of layers made of simple blocks, to form a deep architecture. These blocks are known as neurons, and perform simple functions on their input data, along with an activation function to provide non-linearity. This architecture is then trained using back-propagation to update the coefficients of the individual blocks. There are several deep learning architectures that have been used for the task; Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Attention Mechanism, Graph Convolutional Networks (GCNs). Most approaches utilize a combination of these architectures for modelling the spatial and temporal dependencies.

2.4.1 Modelling Spatial Dependencies

- CNNs (43) (45): Convolutions work by passing a small window of trainable parameters over a grid to give us a new grid which contains some information about local features, as shown in Figure 2.3. This has seen huge success in image processing as it can interpret shapes and textures in local neighbourhoods

of an image (1). In order to use CNNs for traffic analysis, an image of the

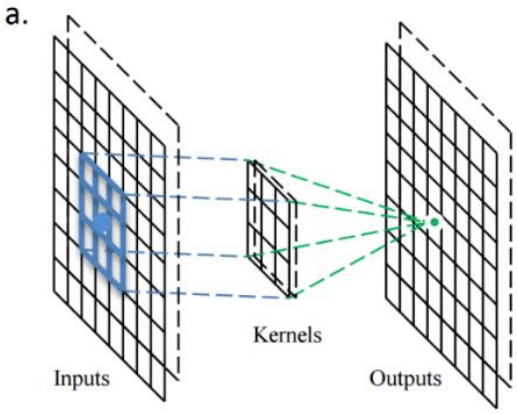


Figure 2.3: Kernel Convolution 2.3

road map is used and divided up into a grid of equal sized rectangles (Figure 2.4). Sensors within each rectangle are aggregated to provide that location's data. This map can then be operated on using 2D convolutions. However, this strategy can be misleading as it uses Euclidean distance between neighbouring cells. In fact, owing to the nature of road networks, sensors right beside each other might take much longer to travel between (e.g. with one way roads) as shown in Figure 2.5.

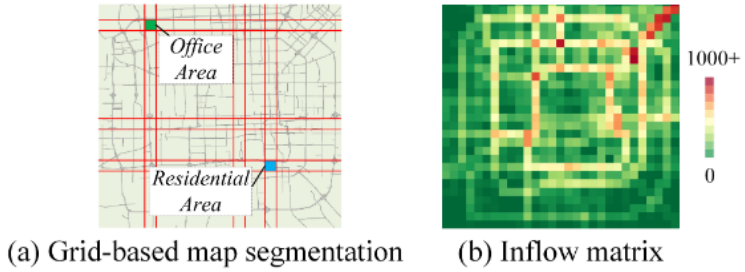


Figure 2.4: Grid based method for CNNs ((42))

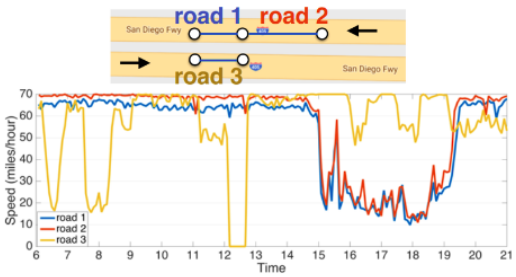


Figure 2.5: Problem with Euclidean Distance as measure in CNNs (26)

- GCNs (18) (12) (27) (17): By converting the road network into a graph structure, we are more accurately able to model its non-Euclidean spatial structure. The convolutions involved differ from those in traditional CNNs as they operate on nodes and their neighbours, rather than on a grid structure. in 2020 DeepMind managed to reduce Google Maps ETA errors by up to 50% (11) using a GCN. I will elaborate further on how GCNs work in the section 2.5.

- Attention (18) (44): The attention mechanism was first developed for Natural Language Processing (2) and works by dynamically assigning different weights to each input. In the context of a road network, that would allow the model to learn the connections between any two nodes without prior information. This allows for many more permutations that could lead to accurate prediction. However, this process requires huge amounts of data and training time, as we are essentially testing over a wide number of adjacency matrices on our nodes.

2.4.2 Modelling Temporal Dependencies

- RNNs (45) (27) (17): RNNs were created for processing sequential data, and involve using identical recurrent block. These blocks take in the current input x_t and the hidden state from the previous block h_{t-1}, and give an output o_t along with the hidden state h_t to be fed into the next block. LSTM (Long Short Term Memory) and GRU (gated Recurrent Unit) are examples of such blocks. The hidden state is very effective at creating and maintaining a *context* that can be used for our predictions.

  A downside of this structure is not being highly parallelisable, as the previous hidden state must be calculated in order to perform the calculation. To combat this, it is common to use an encoder decoder model (as in (27)) where the historical state can be encoded into a fixed length vector. This can then be decoded in order to be used for prediction by future blocks. This can be useful for short input sequences, however above a certain length we begin to lose information.

  RNNs also require more training data since we must train them on a long sequence of data in order to create the context. This means that the entire

sequence cannot be used for testing.

- CNNs (43) (12): CNNs were first used to fully model 'sequence to sequence' (i.e. temporal) dependency by (14). This concept basically involves performing 1D convolutions over time series. This has the benefit over RNNs of being highly parallelisable. Since we can create a feature vector (of previous time steps) for each point we can divide up long sequences and shuffle them into training and test data.

  One method of performing temporal convolutions involves using a dilated causal convolution, which is a specific type of 1D convolution shown in Figure 2.6. In this diagram, our blue nodes would be our input array $x_{t-M}, ..., x_t$,

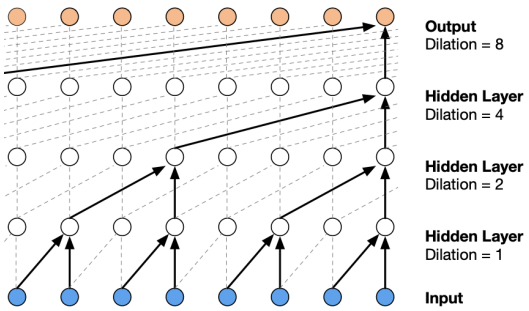


Figure 2.6: Diagram of a dilated causal convolution (37)

  while the orange nodes would represent our outputs $\hat{x}_{t-M+1}, ..., \hat{x}_{t+1}$. These are not necessarily our predictions, as these outputs may be fed into another convolutional layer. The term causal means that the model cannot violate the ordering of the data, in this instance meaning that we can only sample backwards in our convolution. This is shown in the diagram by the fact that all our connections move up or to the right. The term dilated shows that the output receives information from larger and larger timespans as we work down.

- GCNs (33): Using GCNs to model temporal dependency is rare but can be seen in (33). This approach does not use separate modules for spatial and temporal dependencies. Instead, layer our time step graphs on top of each other, with each node at time t linked to its value at $t-1$ and $t+1$. This creates a much larger graph on which a GCN could be used.

- Attention (18) (44): By using attention in the time domain, the model can adaptively select the most relevant historical states. For example, this might mean it learns that the data from 2 days previous is more relevant than 1 day. This is the kind of relation that we would not think to incorporate into the design of other model types. As we saw with using attention for spatial dependencies, this takes huge amounts of data and training.

2.5 Graph Neural Networks

2.5.1 History of GNNs

GNNs are relatively new to the field of Deep Learning, first created by (16). A graph neural network is any neural network that uses graph embeddings to capture the interconnected nature of data points. This can apply to Graph Convolutional Networks (GCNs), Graph Attention Networks (GANs), Graph Recurrent Networks (GRNs), or a combination of these.

Progress in image or language based neural networks are highly publicized as the results are more akin to processes that we as humans perform (such as creating fake images or being able to participate in conversation) making them more understandable as artificial intelligence. Although fascinating, these are just a fraction of the problems we look to address in the world. As such a basic structure, graphs apply to a huge variety of scenarios. In fact, we can even think as images as regular grid graphs, where each pixel is a node connected to 4 others. Some other applications include:

- Protein Structure Prediction (24): AlphaFold 2 is a deep learning algorithm developed by DeepMind to predict a protein's structure from its sequence of amino acids. It made big news in 2020 by winning the CASP (Critical Assessment of Protein Structure) competition, with a score of 92.4%, a score comparable to experimental techniques like X-ray crystallography. This model created graph embeddings from the input data to find the pairwise distances between amino acids in the sequence. From this the structure could be determined.

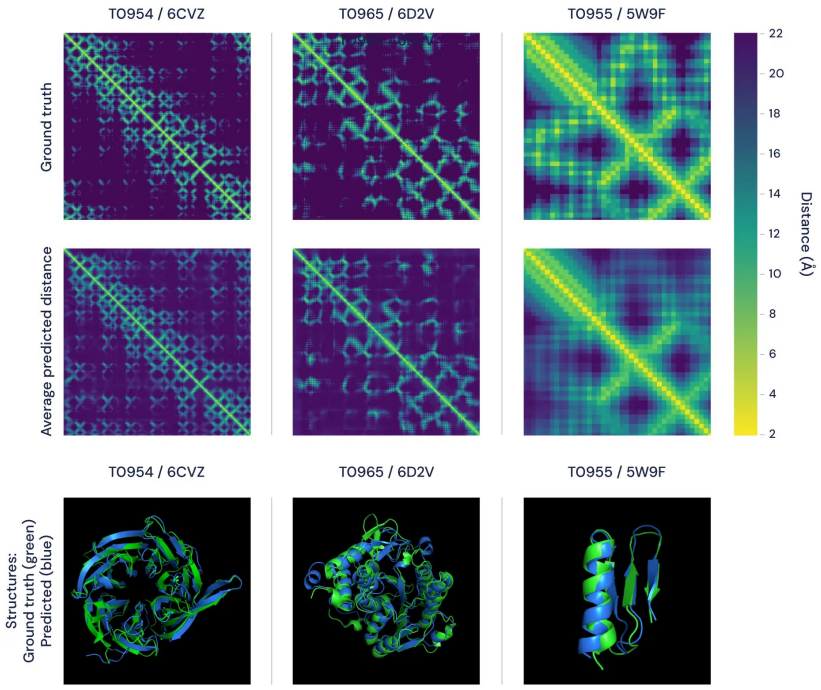


Figure 2.7: Adjacency Matrices in AlphaFold ((24))

- Pollution Forecasting ((38)): The $PM_{2.5}$-GNN model aims to predict pollution, structuring our stations as a spatio-temporal graph. Node attributes like temperature, humidity, and precipitation can vary over time. A more interesting feature is the changing of edge attributes over time, in accordance to the strength and direction of the wind. This means our adjacency matrix is dynamic, and can more accurately represent how stations are connected compared to distance. It then uses a Recurrent structure to predict changes over time.

2.5.2 Spectral Graph Convolutions

Spectral Analysis

Spectral analysis involves studying properties (such as polynomial, eigenvalue, or eigenvectors) of characteristic matrices of graphs. These characteristic matrices of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ (we will denote $|\mathcal{V}|$ as n, the number of nodes):

- Adjacency Matrix A: A square matrix of size $|n| \times |n|$ where the location $A_{i,j}$ marks the edge from node i to node j. In an undirected graph the matrix A will be symmetrical, however this is not the case with a traffic network, where

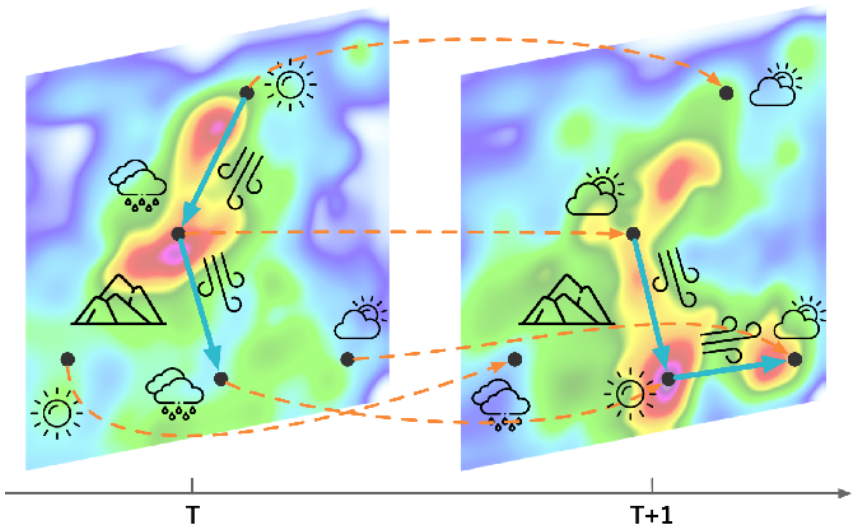


Figure 2.8: Changing Edge Attributes with Time (38)

it may be possible to move from node i to node j but not vice-versa. In a simple (unweighted) graph all values in A will be either 0 or 1. In our case we will add weighting to our graph. Since adjacency is higher for nodes that are more closely linked, $A_{i,j}$ will have an inverse relationship to the travel distance from node i to node j.

- Degree Matrix D: The degree matrix is a diagonal matrix of size $|n| \times |n|$ where $D_{i,i}$ is the degree of node i and all $D_{i,j}$ where $i \neq j$ is 0. The degree of node i is defined by the sum of the weights of all edges that terminate at the node i.

$$D_{i,i} = \sum_{m=0}^{|n|} A_{m,i} \tag{2.1}$$

- Laplacian Matrix L: This is a very widely used graph representation thanks to some useful properties, such as its eigenvalues being non-negative real numbers and its eigenvectors being real and orthogonal.

$$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A} \tag{2.2}$$

- Normalized Adjacency Matrix A_{norm}: Imbalanced weights between rows and columns can lead to instability when processing our matrices. Normalization is a method of scaling our values by the vertex degrees in order to reduce

imbalances.

$$\boldsymbol{A}_{norm} = \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2} \tag{2.3}$$

- Normalized Laplacian Matrix L_{norm}:

$$\boldsymbol{L}_{norm} = \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2} \tag{2.4}$$

- Graph Fourier basis U: The matrix of eigenvectors of L_{norm}.

- Eigenvalue Matrix Λ: A diagonal matrix of the values used to scale the eigenvectors of L_{norm}

$$U\Lambda U^T = \boldsymbol{L}_{norm} = \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2} \tag{2.5}$$

- Identity Matrix I: Simply a diagonal matrix of the same dimensions as our adjacency matrix, with values of 1 along the diagonal. All of these matrices are square matrices of size $|n| \times |n|$ (i.e. they belong to $\mathbb{R}^{|n|\times|n|}$)

Convolution Method

Spectral convolution were initially developed by (3) and uses the characteristic matrices found in spectral analysis. The notation for graph convolutions is usually $\Theta *_{\mathcal{G}} x$, where Θ is our parameters, $x \in \mathbb{R}^n$ is our node value data, and $\mathcal{G}$ is our graph. The general form of convolution defined by (3) was:

$$\Theta *_{\mathcal{G}} x = \Theta L x = \Theta(U\Lambda U^T)x = U\Theta(\Lambda)U^T x \tag{2.6}$$

Approximations

Approximations are often used for the convolution in order to reduce the number of parameters and increase speed. This can be done by first by breaking Θ into

sub-kernels θ_k to multiply with our eigenvectors Λ^k.

$$\Theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k \tag{2.7}$$

The Chebyshev polynomial method (19) uses the Chebyshev polynomial of $\hat{\Lambda}$ as a way of truncating it. $\hat{\Lambda}$ is simply Λ rescaled as $2\Lambda/\lambda_{max} - I$. λ_{max} is the greatest eigenvalue of L

$$\Theta(\Lambda) \approx \sum_{k=0}^{K-1} \theta_k T^k(\hat{\Lambda}) \tag{2.8}$$

$$\Theta *_{\mathcal{G}} x \approx \sum_{k=0}^{K-1} \theta_k T^k(\hat{L}) \tag{2.9}$$

The library ChebNet (10) is often used as a tool for using Chebyshev polynomial approximations in Graph Convolutions.

Generalization

We can then generalize this to signals with many channels, as we have in our case. If we have C_I input channels and C_O output channels our input signal is denoted as $X \in \mathbb{R}^{n \times C_i}$ and our output signal as $Y \in \mathbb{R}^{n \times C_O}$. The parameters are denoted by $\Theta \in \mathbb{R}^{n \times C_I \times C_O}$. Then we calculate each channel of our output as:

$$y_j = \sum_{i=1}^{C_I} \Theta_{i,j}(L) x_i, 1 \leq j \leq C_O \tag{2.10}$$

Using Einstein Summation (ES) notation we can simply denote this as:

$$Y \overset{\text{ES}}{=} \Theta_i(L) X_i \tag{2.11}$$

2.5.3 Spatial Graph Convolutions

Spatial methods were created in (29) and use a more local approach. Rather than transforming the graph into the spectral domain, we instead apply our functions on

individual nodes.We calculate the next-layer node for node v as follows:

$$\mathbf{h}_v^{(k)} = f\left(\mathbf{W}^k\mathbf{x}_o + \sum_{i=1}^{k-1}\sum_{u\in N(0)}\mathbf{\Theta}^k\mathbf{h}_u^{(k-1)}\right) \tag{2.12}$$

f is an activation function here. $\mathbf{W}$ is a matrix of learnable parameters against which we multiply the input vector of node v, $\mathbf{x}_v$. The second half refers to the aggregation of data from surrounding nodes. The internal sum is a sum through all neighbours u of v. The neighbour's hidden state from the previous layer $h_u^{(k-1)}$ is multiplied with our filter matrix $\mathbf{\Theta}^{(k)}$ which is also a set of learnable parameters, as with the spectral method. The outer sum is a summing across channels k. This is only an example of a spatial method, and not the definitive equation.

Spatial methods are more parallelisable and scalable, as they can perform functions on sets of nodes rather than the whole graph at once. They are also able to operate on more complex types of graphs, such as directed graphs. Though the spatial methods are more computationally efficient, the mathematical foundation of the spectral methods gives them deeper rigour.

2.6 STGNNs in Traffic Forecasting

2.6.1 Spatio-Temporal Graph Neural Networks (STGNN)

A spatio-temporal graph is one in which the edges and nodes remain fixed, but the node attributes and edge weights may vary over time.

$$\mathcal{G}(t) = (\mathcal{V}, \mathcal{E}, \mathcal{X}_{\mathcal{V}}(t)) \tag{2.13}$$

An STGNN is any neural network that operates on such a graph. Traffic networks can be thought of as a spatio-temporal graph, with models using sensors as our nodes with incoming values (in the case of using roadside data collection as I am). Edges and their weights can either be constructed from real values (like travel time or distance) in a natural graph, or from by some other factor (like how many routes pass the same two nodes) in a similarity graph. It is even possible to use several overlaid

graphs to encode different kinds of spatial correlations, such as neighbourhood or transportation connectivity (15). A benefit of GNNs is how the graph structure can be very versatile.

In this section, we will use Spatio-Temporal Graph Neural Network (STGNNs) to refer to models that use GCNs to generate spatial dependencies, specifically. This omits (33) which is a special case in which a single GCN models both the spatial and temporal dependencies. All other works that I found used one of Recurrent, Convolutional, or Attention modules to model their temporal dependencies.

The first application of STGNNs in traffic forecasting came in 2017 with (26). This used a spectral encoder-decoder in order to model the spatial dependencies, with Gated Recurrent Units (GRUs) as the recurrent layer in order to model the temporal dependencies. They achieved an improvement of 12-15% over state-of-the-art approaches. Since then there has been a huge uptake in the field with several surveys finding STGNN models to be the most effective (25) (40) (39). Next, I will show examples of STGNNs with each type of temporal modelling.

2.6.2 STGNN with Recurrent Module

Figure 2.9 shows the structure of the Recurrent STGNN used in (27). This model is used to predict traffic speed using Beijing Taxi data. Here a graph convolution is applied to the graph before being entered into the Graph LSTM block. After all $TSG_1, ..., TSG_t$ have been fed into the recurrent block, the hidden state is then encoded into a vector. This allows for more efficient training on smaller data sets, by limiting the size of the hidden state. This is then decoded and used as the hidden state in the next GLSTM block to predict outr output. This can be done to forecast as many steps into the future as we would like, by simply using the resulting hidden state in the next GLSTM again and again.

Another example is (17) which was used to predict both traffic speed on Washington DC data and flow on the PeMSD4 dataset, demonstrating the versatility of this approach. This paper used GRU models as their recurrent block.

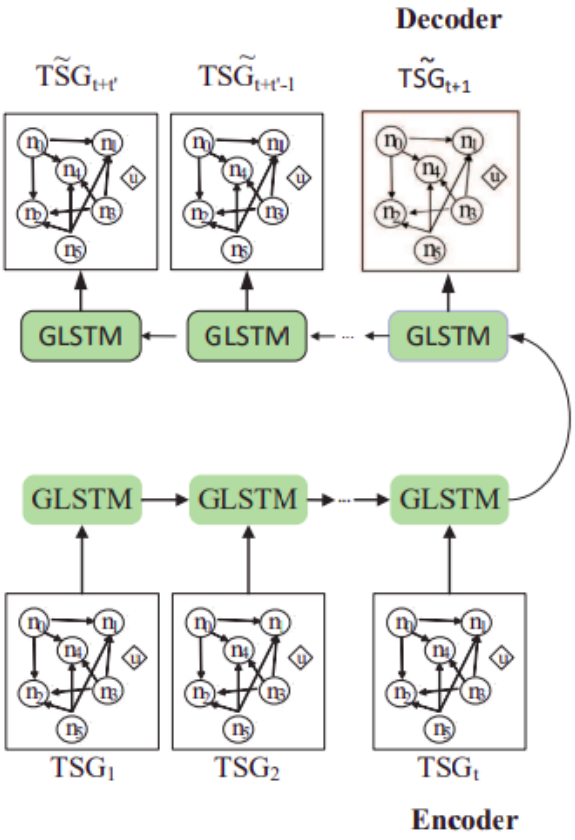


Fig. 4. Framework of the GLSTM model.

Figure 2.9: Example structure of a Recurrent Graph Neural Network (27)

2.6.3 STGNN with Convolutional Module

Figure 2.10 shows the structure of (41). This particular model is used to predict traffic flow using the BJER4 and PeMSD7 datasets.

The ST-Conv Block contains both the spatial and temporal convolutions. Inside each temporal block we are performing 1D convolutions on each node's signal. We do 64 of these convolutions to get 64 output channels for increased complexity. Next, the Spatial Graph-Conv Block aggregates information from these 64 channels across our nodes and outputs 16 channels. Finally, this new information is passed through a Temporal Block again, before finally being linearized in the final layer, to make our predictions

I have based my model on this paper, and so I will describe the structure further in section 3.

Another example of a Convolutional STGNN is (12), though this uses a slightly

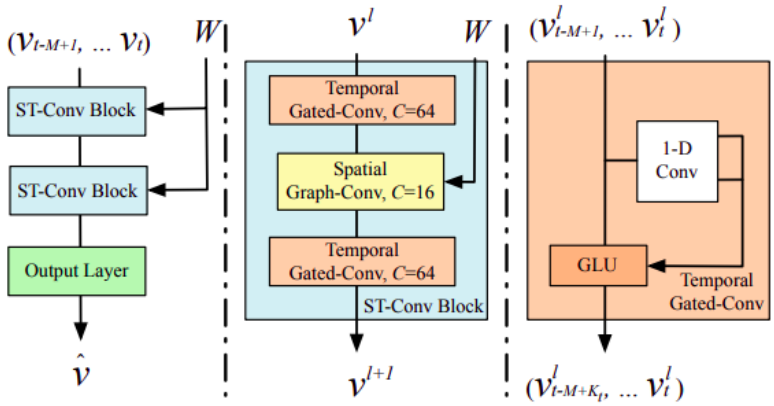


Figure 2.10: Example structure of a Convolutional STGNN (41)

different configuration for its spatial module. It involves two parts for both local (relating to a node's neighbour on a natural graph) and global predictions (information is accrued from further across the graph).

2.6.4 STGNN with Attention Module

Figure 2.11 shows the attention based STGNN used in (18). Here the three identical

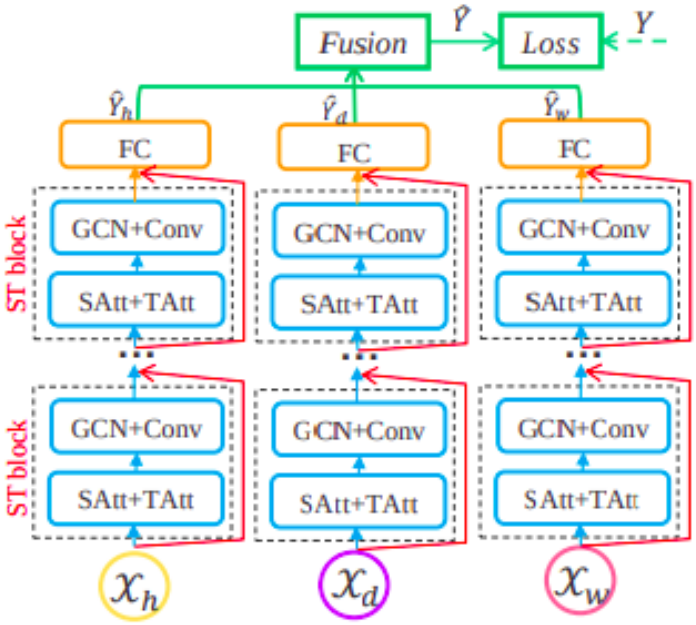


Figure 2.11: Example structure of an Attention based STGNN (18)

structures are used to model different aspects of the data. Each structure is provided with a different set of previous time steps; one showing the preceding hours x_h, one showing this hour on previous days x_d, and one showing this hour in preceding

weeks x_w. Each structure consists of two Spatio-Temporal Blocks followed by a fully connected layer. Each ST Block is very complex with spatial and temporal attention modules followed by a GCN and 1-D convolution block. The attention modules allow nodes to find patterns in nodes that were unconnected on our input graph. This uses many more trainable parameters than Recurrent or Convolutional STGNNs and takes much longer to train/

2.7 Chosen Model

I decide to use the STGCN model (41) for predicting traffic flow in Dublin City. I chose this model for two reasons. Firstly, it models the spatial dependencies with spectral graph convolutions, which have been shown to achieve high performance on traffic networks. The graph structure describes our road network better than grid convolutions seen in other models, and require less training data and time than attention based methods. Although spectral methods are more computationally expensive than spatial graph convolutions, they are more robust and do not require as much fine-tuning in our hyperparameters. STGCN uses 1D convolutions to model temporal dependencies. These are more parallelisable than recurrent methods as we can make forecast based on a feature vector rather than passing over many previous time steps to develop the hidden state. The feature vectors also allow us to completely shuffle our training and test data, which is not the case for the recurrent method where we must divide our data into long training and testing segments. 1D convolutions require a lot less data and training time than attention based methods.

Chapter 3

Methodology

Firstly, I will describe how the data is collected and preprocessed for use in my model. I will then describe, in detail, the inner structure of the STGCN model, including the function of each part. I will then look at how my model is trained and performance metrics used. I also describe a different version of the model in which an estimate distribution is created. Finally, I describe the baseline model that I will use for comparison in my experiments.

3.1 Data Collection

3.1.1 SCATS

In this project, I will be working with the Dublin City SCATS (Sydney Coordinated Adaptive Traffic System) data set. This collects data from 676 locations from the Greater Dublin Area (7). The distribution of these junctions is shown in Figure 3.1, with darker regions containing more junctions. This is plotted using Uber's H3 spatial index ((20)).

The system updates after each cycle and provides data on each lane of traffic entering a junction. The data provided includes:

- PT: The phase time, or duration of each phase of the traffic light cycle
- VO: The volume, or number of vehicles that pass in that phase.
- DS: The degree of saturation, or extent to which a lane is used. Calculating

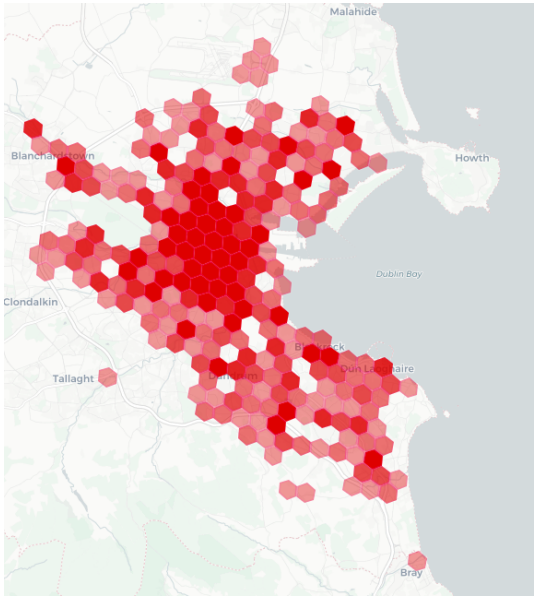


Figure 3.1: Distribution of SCATS monitored junctions in Dublin City

density from this is shown in (9).

From these metrics we can calculate the flow and density of my traffic.

3.1.2 Interpreting Junctions

The first step is using the SCATS data along with the Dublin City Council maps to understand which lanes in my text file correspond to which phases in my text file. It is important to understand my junctions in order to create my adjacencies later on. We can do this by looking at the phase name in my text file (though this can be misleading) and the number of lanes. Figure 3.2 shows a diagram of a junction and Figure 3.3 shows the corresponding text file. SCATS Traffic Reporter is used to generate these text files from the SV files that are outputted by the SCATS software.

3.1.3 Text File Parser

The text file is outputted per day by entering the date and subsystem numbers of the junctions we wish to analyse. The parser is seen in the files `nv_from_txt` files in my `interpret_csv` folders. The parsers are slightly different for each junction subset.

From Figure 3.3 we can see that the text requires some preprocessing before being used as my data. First I replace all {*!'>^} characters with spaces as these symbols

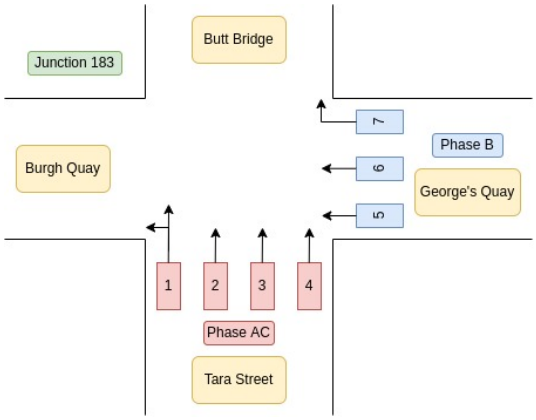


Figure 3.2: Junction 183 Diagram

```
Strategic Monitor On Sunday 01-April-2018 filename:CCITY_20180401

Sunday 01-April-2018 00:01 SS   6M+ PL 6.3  PV   7.3 CL  118 +0 RL  94' SA 20  DS 47
 Int   SA/LK   PH  PT!  DS  VO  VK!  DS  VO  VK!  DS  VO  VK!  DS  VO  VK! ADS
  183 S  20  ' AC  75!  47  10  11!  27   8   7!  25   9   7!   9   3   2!   43
  183 S  21 ^'  B  27!  10   2   1!  32   4   3!  32   6   3!   -       -!   38
  184 S 161 *   A 111!  31  14  11!  45  19  17!  31  15  13!   8   3   3!   40
  184 S 162 *   B   0!   0   0   0!   0   0   0!   -       -!   -       -!    4
  183 S 183 *   B  27!  10   2   1!  32   4   3!   -       -!   -       -!   38
  183 L   7  ' AC  75!  47  10  11!   -       -!   -       -!   -       -!  320
A=<35>  B=48  C=#20
Sunday 01-April-2018 00:01 SS  10M+ PL 4.2  PV   4.2 CL  118 +0 RL  93' SA 20  DS 47
 Int   SA/LK   PH  PT!  DS  VO  VK!  DS  VO  VK!  DS  VO  VK!  DS  VO  VK! ADS
  354 S  30  '  A  92!  17   8   5!  17   8   5!  11   5   3!   -       -!   20
  354 S  31 ^   B  20!  22   4   2!  22   3   2!  12   2   1!   -       -!   23
  182 S  34 ^   A  97!   0   0   0!  12   4   4!   -       -!   -       -!   16
    2 L  14  ' AD  94!  31  12   9!  35  15  14!   -       -!   -       -! 1360
    2 L  15  ' AD  94!   -       -!   -       -!   -       -!   -       -!    0
A=<47>  B=53
Sunday 01-April-2018 00:01 SS  58M+ PL 4.3  PV   4.3 CL  118 +2 RL  93' SA 0   DS 0
 Int   SA/LK   PH  PT!  DS  VO  VK!  DS  VO  VK!  DS  VO  VK!  DS  VO  VK! ADS
  288 S 206     A  86!  38  14  12!  16   7   4!  39  14  12!   0   0   0!   34
  288 S 270 ^   B  13!   0   0   0!  32   3   1!   -       -!   -       -!   47
A=<76>  B=#28
Sunday 01-April-2018 00:01 SS  85M+ PL 5.3  PV   6.3 CL  120 +0 RL 108' SA 313 DS 153
 Int   SA/LK   PH  PT!  DS  VO  VK!  DS  VO  VK!  DS  VO  VK!  DS  VO  VK! ADS
  908 S 287 ^' AC  83!   -       -!  28  10   7!  18   7   4!   -       -!   29
  196 S 310  '  A  57!  71   9  12!  65   6  10!  76   6  13!   -       -!   82
  196 S 311  'ABE  57!  11   3   2!  44   7   8!  62  10  12!  22   4   5!   68
  196 S 312 ^'  D  48!  47   6   6!  39   7   6!  72  12  13!  57   7   8!   68
  197 L  71  'ABF  40!  69   9  11!  51   9   9!   -       -!   -       -!  520
  197 L  72  'ABF  40!  54   7   7!  64  11  10!   -       -!   -       -!  440
A=<18>  B=20  C=#25  D=24  E=17
Sunday 01-April-2018 00:03 SS   6M+ PL 6.3  PV   6.3 CL  116 +0 RL  97' SA 20  DS 48
 Int   SA/LK   PH  PT!  DS  VO  VK!  DS  VO  VK!  DS  VO  VK!  DS  VO  VK! ADS
  183 S  20  ' AC  82!  48  11  12!  27   9   7!  45  16  14!   8   3   2!   47
  183 S  21 ^'  B  41!  77  13  10!  67  10   9!  32   5   4!   -       -!   54
  184 S 161 *   A 108!  36  13  13!  50  17  19!  34  14  14!  13   5   5!   45
  184 S 162 *   B   0!   0   0   0!   0   0   0!   -       -!   -       -!    0
  183 S 183 *   B  41!  77  13  10!  67  10   9!   -       -!   -       -!   54
  183 L   7  ' AC  82!  48  11  12!   -       -!   -       -!   -       -!  360
A=<35>  B=48  C=#20
Sunday 01-April-2018 00:03 SS  10M+ PL 3.2  PV   3.2 CL  116 +0 RL  95' SA 20  DS 48
 Int   SA/LK   PH  PT!  DS  VO  VK!  DS  VO  VK!  DS  VO  VK!  DS  VO  VK! ADS
  354 S  30  '  A  93!  25   9   8!  22   8   7!  19   7   5!   -       -!   21
  354 S  31 ^'  B  24!   7   2   1!  33   5   3!  11   2   1!   -       -!   28
  182 S  34 ^   A  75!   0   0   0!   0   0   0!   -       -!   -       -!    7
    2 L  14  ' AD  74!  27   8   6!  41  14  13!   -       -!   -       -! 1400
    2 L  15  ' AD  74!   -       -!   -       -!   -       -!   -       -!    0
A=<53>  B=47
```

Figure 3.3: Text File containing Junction 183 data for 01/04/2018

are not relevant to my use. Next I replace unused lanes {- -} with {- - -}. Finally, I substitute any length of whitespace with a comma. This means we are able to save the file in CSV format.

We can then read this CSV file into a Pandas Dataframe, with columns "Day", "INT", "OTHER", "SA/LK", "PH", "PT", "DS_A", "VO_A", "VK_A", "DS_B", "VO_B", "VK_B", "DS_C", "VO_C", "VK_C", "DS_D", "VO_D", "VK_D",

"ADS". I then iterate through each row of this Dataframe to find the appropriate information.

Each time I reach a metadata line (containing the date, time, and junction information) I update my current time. This current time is used later to find the index this data will take in my tensor. We can tell we are at a metadata line if the OTHER column in the CSV file.

Due to the layout of the file, there are a few checks we need to do in order to establish if a row has relevant data in it.

- The SA/LK column must have a numerical value. This distinguishes my data rows from my metadata rows.

- The INT column must contain the number of one of my junctions. In some cases, the file will include data for junctions other than the ones I am using.

- Do not include duplicate data. This requires more handcrafting as some junctions output duplicates of their data and this can only be seen by looking at whether that phase has already been read or by an incorrect number of lanes. We can see this in Figure 3.3, where the data for junction 183 is printed, followed by the data for junction 184, followed by a repeat of some 183 data.

For each data row I add the calculated variables of flow and density before using. `nv_from_txt` iterates through each text file in my data folder.

3.1.4 Calculating Variables

On each data row, I must calculate the flow (Q) and density (K). I sum the values in my `VO_x` columns (lanes) to find the estimated vehicles per phase (VO) on my node. I average across my `DS_x` columns to find the degree of saturation (DS) of my road segment. I also read the `PT` column to find the phase time (PT).

We calculate the flow (in vehicles per hour) using the equation:

$$Q = \frac{VO * 60 * 60}{PT} \tag{3.1}$$

We then calculate the density using:

$$K = \frac{1 - ((1 - DS) * PT + t * VO)}{PT} \tag{3.2}$$

These equations are found in (8).

3.1.5 Creating the Tensor

We create 2 arrays per day; one for flow and one for density. They are of dimensions $[T_d \times N]$ (where T_d is the number of intervals in a day and N is the number of nodes in my graph) and are initialized with value `NaN`. This again is done in the `nv_from_txt` files.

The text file gives updates on each junction every 1 to 2 minutes, so I am using intervals of 3 minutes in order to capture a stable signal. This gives use $24 * 60/3 = 480$ intervals per day.

Once I have calculated the flow and density on a row of data, I find their sensor index by applying the `sensor_to_index` function. This takes the junction number, phase, and number of lanes to find which node the data corresponds to. I find their interval index using the `time_to_index` function. This takes the hours h and minutes m to give us the corresponding index $\lfloor$h*(60/3) + m/3$\rfloor$. If there is already a value stored in the array for that set of indices, I take the average of the old and new value. This will happen when a junction outputs data more than once in a 3-minute period.

After each day file is processed, I perform the `fill_array` function. This checks for empty cells and interpolates data along the time interval axis if they are found. Next, each array is concatenated to the end of a running array for each metric, with dimensions $[m * T_d \times N]$. m is the number of days I have read in so far with $m * T_d = T_{total}$ once all days have been added.

Once all days have been read in I have two arrays of dimensions $[T_{total} \times N]$ which I stack to make a tensor of dimensions $[T_{total} \times N \times C]$. C is the number of channels, which is two in my case. This tensor will next be used as an input to my `load_data` function.

3.1.6 Creating Distance Matrix

I used OpenStreetMap API to create my distance matrix. First, I found the coordinates of each of my sensors. It is important that these are sensor specific, and not just one for the junction, as some incoming routes to the junction cannot leave in the same way as others. I then perform an outer loop of origin nodes, and an inner loop of destination nodes loops.

For each combination, I use the API to find the directions from the origin to the destination and take the distance in kilometres. When the origin and destination are the same, the distance is zero. This gives us a distance matrix of dimensions $[N \times N]$, an example of which is shown in Figure 3.4.

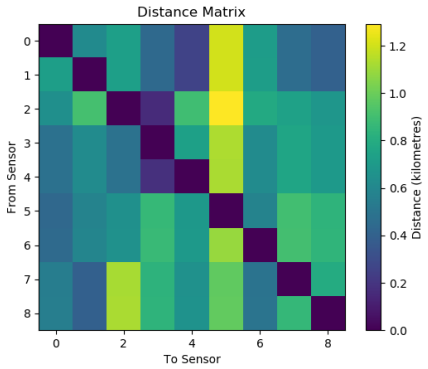


Figure 3.4: Example of a distance matrix

3.2 Data Pre-Processing

3.2.1 Pruning Distance Matrix

We can change some elements of my distance array manually in order to more accurately represent the traffic network. In cases where I do not think a route would realistically be taken from one node to another, I can set that distance as infinite. The most basic version of this would be between sensors at the same junction as this would require a driver to double back on themselves. A more detailed version would involve looking closely at which origins and destinations would be realistic in travelling around the city. The actual decision of how to best prune my matrix is discussed in Section 4.

3.2.2 Kernelization

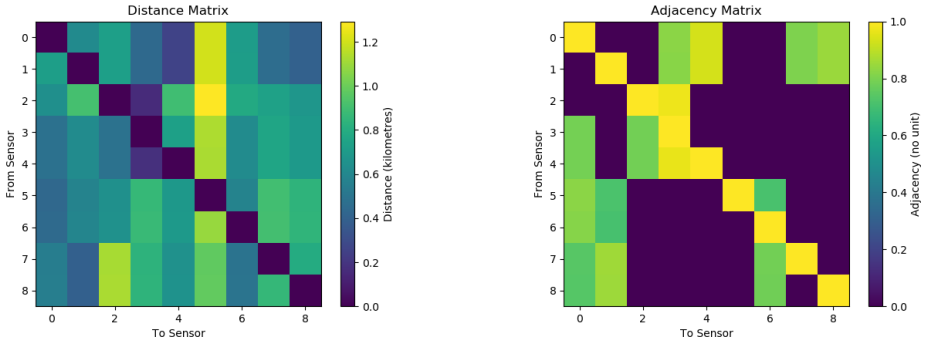
To create an adjacency matrix from a distance matrix I must use some function that gives adjacency an inverse relationship with distance. One method (used in (28), (41)) is to use a Gaussian kernel on each value of my distance matrix for this purpose. This function is shown below with my distance matrix d, adjacency matrix a, and two parameters ϵ and δ^2.

$$a_{i,j} = \begin{cases} \exp\left(-\frac{d_{i,j}^2}{\delta^2}\right) & d \leq \epsilon \\ 0 & d > \epsilon \end{cases} \tag{3.3}$$

ϵ controls the sparsity of the matrix. If two sensors are more than ϵ kilometres apart I consider them unconnected in my graph. Sparser graphs can lead to faster training.

δ^2 dictates the spread of values in my non-zero elements. From the above equation, we can see that the minimum non-zero value for $a_{i,j}$ is $\exp\left(-\frac{\epsilon^2}{\delta^2}\right)$ when $d = \epsilon$. The maximum value for $a_{i,j}$ is one, when $d_{i,j}$ is zero (which occurs when $i = j$).

The results for kernelization can be seen in Figure 3.5,on an un-pruned distance matrix. This example uses values of $\epsilon = 0.6$ and $\delta^2 = 1$ for demonstration purposes.



(a) Distance Matrix (b) Adjacency Matrix

Figure 3.5: Converting a Distance Matrix into an Adjacency Matrix using a Gaussian Kernel

3.2.3 Node Value Normalization

Normalization is normally done on data being use with neural networks. Having data centred around 0 and scaled to a standard range improves training perfor-

mance. Normalized data means that my parameter space is also normalized and my optimization algorithm can search for optimal parameters more easily.

The normalization of the node values is done in the `load_scats_data` function. I find the mean (μ_C) and standard deviation (δ_C) of all values in each channel C of my data tensor. I then normalize using the z-score formula:

$$\hat{X}_C = \frac{X_C - \mu_C}{\delta_C} \tag{3.4}$$

This is for training purposes. When using the model for predictions, I de-normalize the output by performing:

$$X_C = \hat{X_C}\delta_C + \mu_C \tag{3.5}$$

The `load_scats_data` function outputs the means and standard deviations of the channels for this purpose. The de-normalization process is done in the `evaluate_model` file.

This means that the loss values in my training curves are not representative of the loss of my true predictions, as the training losses operate on $\hat{X}_C$.

3.2.4 Adjacency Matrix Normalization

The adjacency matrix is normalized in the function `get_normalized_adj`. This is a different process to statistical normalization and is instead the spectral normalization described in 2.5. This involved scaling my adjacency matrix A by my degree matrix D:

$$\boldsymbol{A}_{norm} = \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2} \tag{3.6}$$

Similarly to the node value normalization, this makes training more stable.

3.2.5 Creating Feature Vectors

The feature vectors are created in the `generate_feature_vects` function (in `utils.py`. This takes the $[T_{total} \times N \times C]$ tensor from `generate_dataset` and converts it into 2 tensors; the inputs and the targets.

For each interval I will create a feature vector of previous intervals used for my predictions (of length F_I) and a target vector of future intervals I wish to predict for (of length F_O). I will take d_p as the distance to the furthest interval back that I use in my feature vector, and d_f as that of the furthest interval forward in my target vector.

The number of usable samples is constricted by $d_p + d_f$ as samples within d_p of the start or d_f of the end of the dataset will not be able to fill their vectors. By appending my feature vectors on each usable interval I get an input tensor of dimensions $[T_{total} - (d_p + d_f) \times N \times C_I \times F_I]$. By doing the same with my target vector I get the target tensor $[T_{total} - (d_p + d_f) \times N \times C_O \times F_O]$.

The values for the input vector were chosen using autocorrelation analyses of my junctions. The autocorrelation plot for one of my junction subsets is shown in Figure 3.6. Here t is the current interval, meaning that $t + 5$ is the interval we are tying to predict. The value of 0 on the x-axis represent $t + 5$.

The plot for each junction varies greatly, but they follow the same general pattern. There are small scale peaks at $n * 480 - 5$ intervals denoting whole days before the data point being predicted. The size of these peaks is also cyclical, with the largest being at $n * (480 * 7) - 5$ which corresponds to this time an integer number of weeks previous. The autocorrelation plots will be discussed further in Section 4.

Table 3.1 shows the values I used in the feature vectors. I used 12 intervals (48 minutes) directly previous to my t, along with 5 intervals (15 minutes) leading up to the $t - (480) + 5$ (the interval exactly one day before my target $t + 5$) and 5 intervals (15 minutes) leading up to the $t - (480 * 7) + 5$ (the interval exactly one week before my target $t + 5$). In this case $d_p = (480 * 7) = 3360$. My feature vector is only $t + 5$ making $d_f = 5$. However, the target vector allows for us to change what I am predicting. I could use a different interval (e.g. one hour ahead $t + 20$) or a range of values (e.g. each interval up to 15 minutes ahead $t + 1, t + 2, t + 3, t + 4, t + 5$)

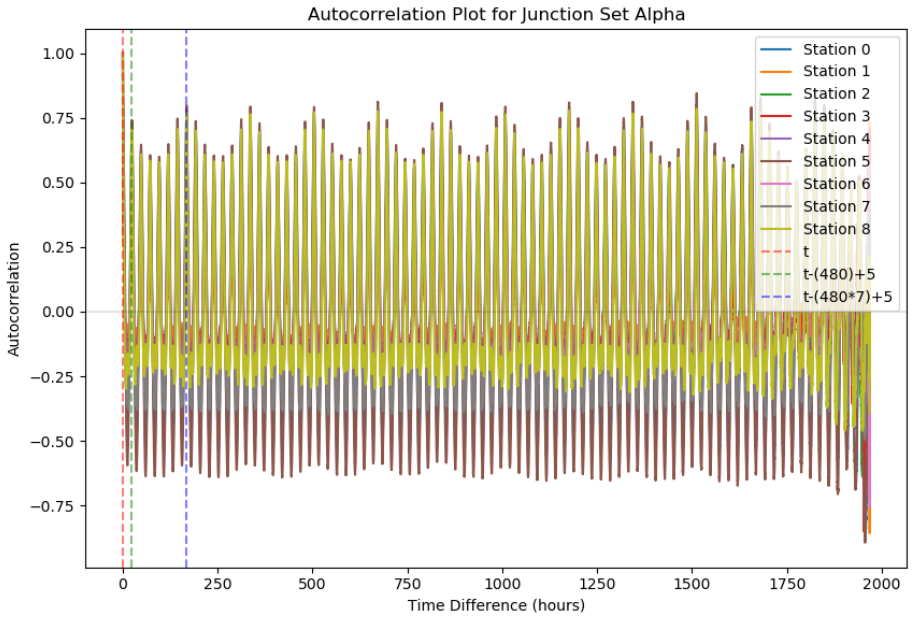


Figure 3.6: Autocorrelation Plot for Alpha Junctions

In my case I am using 2 input channels (flow and density) and one output channel (flow). This means my input tensor has shape $[T_{total} - (3365) \times N \times 2 \times 25]$ and my output tensor has shape $[T_{total} - (3365) \times N \times 1 \times 1]$.

Feature Vector Index	Time Interval
0	$t-(480*7)$
1	$t-(480*7)+1$
2	$t-(480*7)+2$
3	$t-(480*7)+3$
4	$t-(480*7)+4$
5	$t-(480*7)+5$
6	$t-(480)$
7	$t-(480)+1$
8	$t-(480)+2$
9	$t-(480)+3$
10	$t-(480)+4$
11	$t-(480)+5$
12	$t-12$
13	$t-11$
14	$t-10$
15	$t-9$
16	$t-8$
17	$t-7$
18	$t-6$
19	$t-5$
20	$t-4$
21	$t-3$
22	$t-2$
23	$t-1$
24	t

Table 3.1: Input Feature Vector

3.3 Model Structure

The model I am using is based on the original STGCN model introduced by Yu, Yin, and Zhu (41). This was briefly described in Section 2.6.3. This model was used for traffic prediction on BERJ4 and PeMSD7 datasets. They use flow only as their node value. The basic structure for the STGCN modules can be found on GitHub. I have made changes to the structure in terms of order and number of modules used, input channels, and including Dropout and Batch Normalization. I define my structure in the `stgcn` file.

3.3.1 Dimensions

- N: The number of nodes (traffic sensors) in my graph.
- B: Batch size. In training is the number of samples being entered into the

network in one batch in training. In testing this is the number of samples that I wish to predict for.

- F: Number of intervals in my feature or target dimension. F_I indicates the length of the input feature vector, which is 25 in my case. F_O length of my output vector, which here is 1.The size of my feature vector is trimmed by 2 with each Time Block due to the nature of convolutions.

- C: Number of channels: This is the number of node values at each time step. C_I represents my input channels and is set as 2 (although this will be re-examined in Section 4.10). C_O is the number of output channels, which here is only 1. The number of channels is expanded greatly within the STGCN Blocks to allow us to model greater complexity.

3.3.2 Overall Structure

At the top level, there are 3 components to my model; two STGCN Blocks and a Linear layer. This structure is shown in Figure 3.7. In my diagrams I use red to show blocks with internal structure, green for layers with no internal structure, yellow for non-layer processes, and blue for my inputs and outputs.

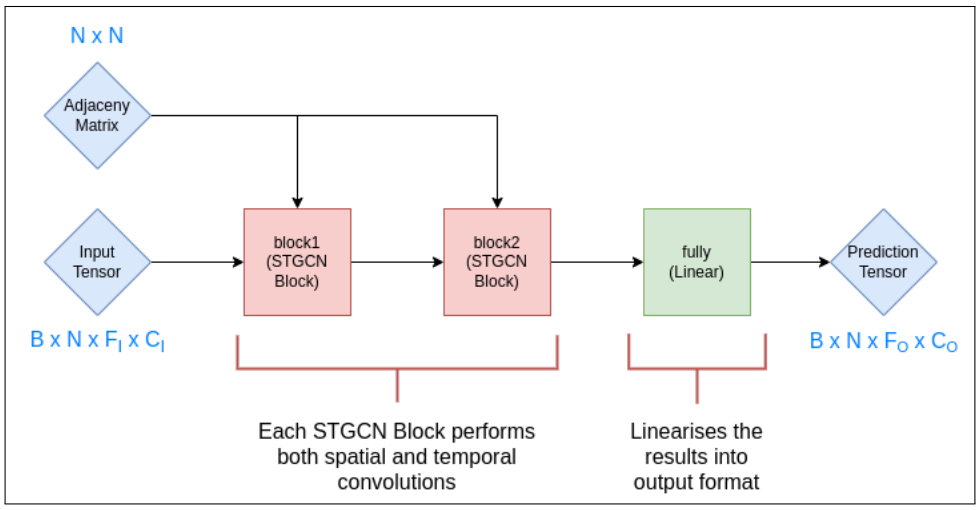


Figure 3.7: Overall DC-STGCN structure

`block1`

Each STGCN block contains both spatial and temporal convolutions. `block1` takes my initial input tensor of size $[B \times N \times F_I \times C_I]$ and my adjacency matrix of size $[N \times N]$ as inputs.

It takes parameters of `in_channels`=C_I, `out_channels`=64, `spatial_channels`=16, `num_nodes`=N. These parameters will be explained in Section 3.3.3.

The output is a tensor of size $[B \times N \times F_I - 4 \times 64]$.

`block2`

Takes the output from `block1` along with my adjacency matrix and performs the spatial and temporal convolutions again.

It uses parameters of `in_channels`=64, `out_channels`=64, `spatial_channels`=16, `num_nodes`=N.

The output tensor is of size $[B \times N \times F_I - 8 \times 64]$.

`fully`

The `fully` block is a fully connected layer. This means that I specify the size of my output tensor, and value of this tensor is created through a linear combination of all values in the input tensor. I use this to reduce my feature vector dimension to the desired target size F_O, and the number of channels to C_O. The final output is then of size $[B \times N \times F_O \times C_O]$.

3.3.3 STGCN Block

STGCN Blocks are initialised with parameters `in_channels`, `out_channels`, `spatial_channels`, and `num_nodes`(=N always). I will use l to refer to a block index, with the subscript of l relating to the input of the block and the subscript of $l+1$ relating to the output. `in_channels` to represent the number of input channels C_l which is not necessarily 2, as is the case with my initial feature tensor, since `block2` is operating on the output of `block1` with 64 channels. `output_channels` is the number of channels in the output of the STGCN Block. `spatial_channels` is the number of channels C_S

I output from my spatial convolution. The structure of the STGCN block is shown in Figure 3.8.

Mathematical Notation

The mathematical notation for my STGCN block is shown below. $*_{\mathcal{T}}$ and $*_{\mathcal{G}}$ are my temporal and spatial convolutions respectively, which will be described in more detail later in this section. M_l is the tensor inputted into my STGCN Block l and M_{l+1} is the resulting output. $\Gamma_{l,0}$ and $\Gamma_{l,1}$ relate to the sets of kernels used in my temporal convolutions, while Θ_l is the parameter tensor used in the spatial convolution.

$$M_{l+1} = \Gamma_{l,1} *_{\mathcal{T}} \text{ReLU}\left(\Theta_l *_{\mathcal{G}} \left(\Gamma_{l,0} *_{\mathcal{T}} M_l\right)\right) \tag{3.7}$$

`temporal1`

The Temporal Block performs convolutions on each node separately, without utilizing spatial information. The block reduces the feature vector length by 2. The input is in the shape $[B \times N \times F_l \times C_l]$ and the output is in the form $[B \times N \times F_l - 2 \times C_{l+1}]$. I will call this output $M_{\mathcal{T}_1}$ for further explanation.

Matrix Multiplication 1

The matrix multiplication is where the adjacency matrix is introduced. The output tensor from `temporal1` is multiplied with my adjacency matrix $[N \times N]$. The output $A_{norm}M_{\mathcal{T}_1}$ has the same shape as my input $[B \times N \times F_l - 2 \times C_{l+1}]$.

Einstein Summation

The Einstein Summation, following the matrix multiplication, completes my graph convolution, as described in Section 2.5. I introduce $\Theta_l \in \mathbb{R}^{N \times C_O \times C_S}$ as my spatial parameters here.

$$\Theta_l *_{\mathcal{G}} M_{\mathcal{T}_1} = \sum_{i=1}^{C_O} \Theta_{i,j}(A_{norm})m_{\mathcal{T}_1,i}, \quad 1 \leq j \leq C_S \tag{3.8}$$

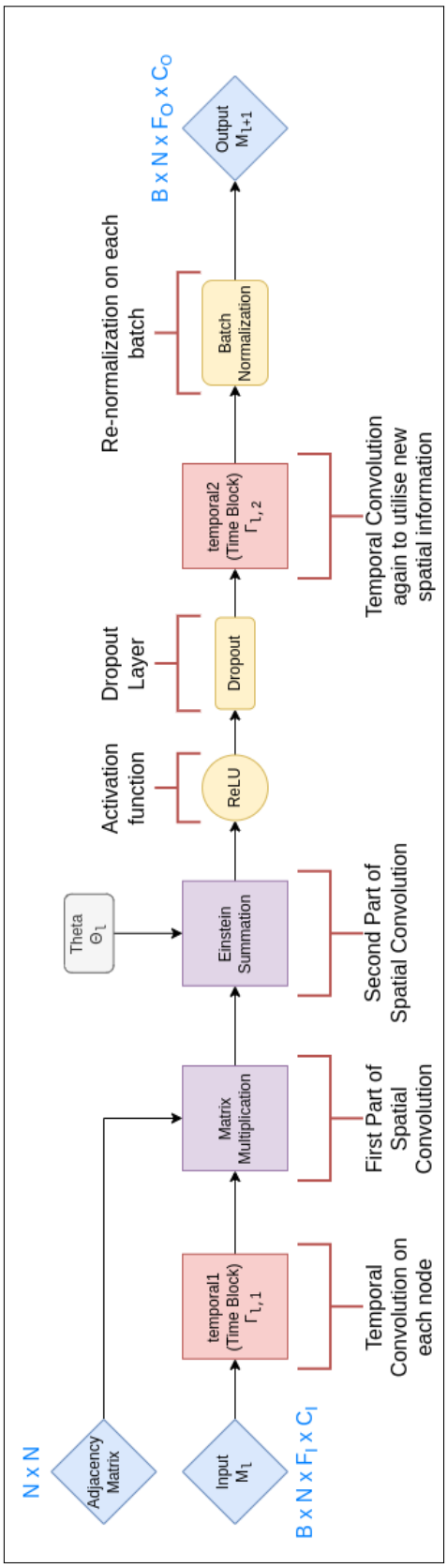


Figure 3.8: Structure of my STGCN Block

$m_{\mathcal{T}_1,i}$ corresponds to channel i of tensor $M_{\mathcal{T}_1}$. With Einstein Summation notation this is written as

$$\Theta_l *_{\mathcal{G}} M_{\mathcal{T}_1} \overset{\mathsf{ES}}{=} \Theta_l A_{norm} M_{\mathcal{T}_1,i} \tag{3.9}$$

The characteristic matrix A_{norm} is used in the STGCN paper (41), rather than the L_{norm} as used in the first definition of a graph convolution (3). This is equivalent since the only difference between them is the subtraction of I which carries no information (an isomorphic transformation). This will just lead to the training of different parameters Θ.

$$A_{norm} = I - L_{norm} = D^{-1/2} A D^{-1/2} \tag{3.10}$$

The output of my graph convolution has dimensions $[B \times N \times F_l - 2 \times C_S]$.

ReLU

ReLU (REctified Linear Unit) is an activation function that is applied to all values in my tensor. Activation functions are what provides non-linearity to neural networks, allowing complexity greater than matrix multiplication. Universal approximation theorem shows that with sufficient width or depth a neural network that uses activation functions can model any function. ReLU is simply defined as:

$$\text{ReLU}(x) = x^+ = \max(0, x) \tag{3.11}$$

The output of my ReLU is of the same dimensions as its input since it operates element-wise.

Dropout

Dropout is a method of regularization to prevent overfitting in my model training. I simply set a fraction of the values in my tensor to 0. By doing this the model becomes more robust, and learns not to rely too much on any one part of the tensor. I used a dropout rate of 0.3, meaning 30% of values are set to zero. This is deactivated in

validation and testing.

`temporal2`

This applies another temporal convolution on each of my nodes. After the spatial convolution has been performed for each time step we want to propagate these new insights along my time axis. The input for this block has the same shape as the output of the spatial convolution $[B \times N \times F_l - 2 \times C_S]$. The number of channels is changed to C_O and the feature length is cut down again by 2 to give the dimensions $[B \times N \times F_l - 4 \times C_O]$, which is the shape of my output M_{l+1}.

Batch Normalization

Each batch-channel slice of my tensor is normalized using the formula:

$$y_{b,c} = \frac{x_{b,c} - \mu_{b,c}}{\sqrt{\sigma^2_{b,c}}}.\gamma + \beta \tag{3.12}$$

The subscript b, c relates to a specific batch-channel and γ and β are trainable parameters.

3.3.4 Time Block

The Time Block performs my temporal convolutions using 1D convolutions (with F as the dimension in question). The structure is shown in Figure 3.9. The subscript p indicates inputs and parameters of p, while $p+1$ indicates the output of the block. The Time Block is initialized with parameters `in_channels` C_p, `out_channels` C_{p+1}, and `kernel_size` K.

Mathematical Notation

We will use p to denote the Time Block index (two numbers in my overall structure as there are two layers of abstraction we must specify). M represents my data tensor. Γ_p, x is the specific trainable kernel in 1D convolution x, whereas Γ_p represents all kernels used in the block. $\boldsymbol{\sigma}$ is my sigmoid activation function.

$$M_{p+1} = \Gamma_p *_{\mathcal{T}} M_p = \text{ReLU}\left(\Gamma_{p,1} *_{1D} M_p + \boldsymbol{\sigma}\left(\Gamma_{p,2} *_{1D} M_p\right) + \Gamma_{p,3} *_{1D} M_p\right) \tag{3.13}$$

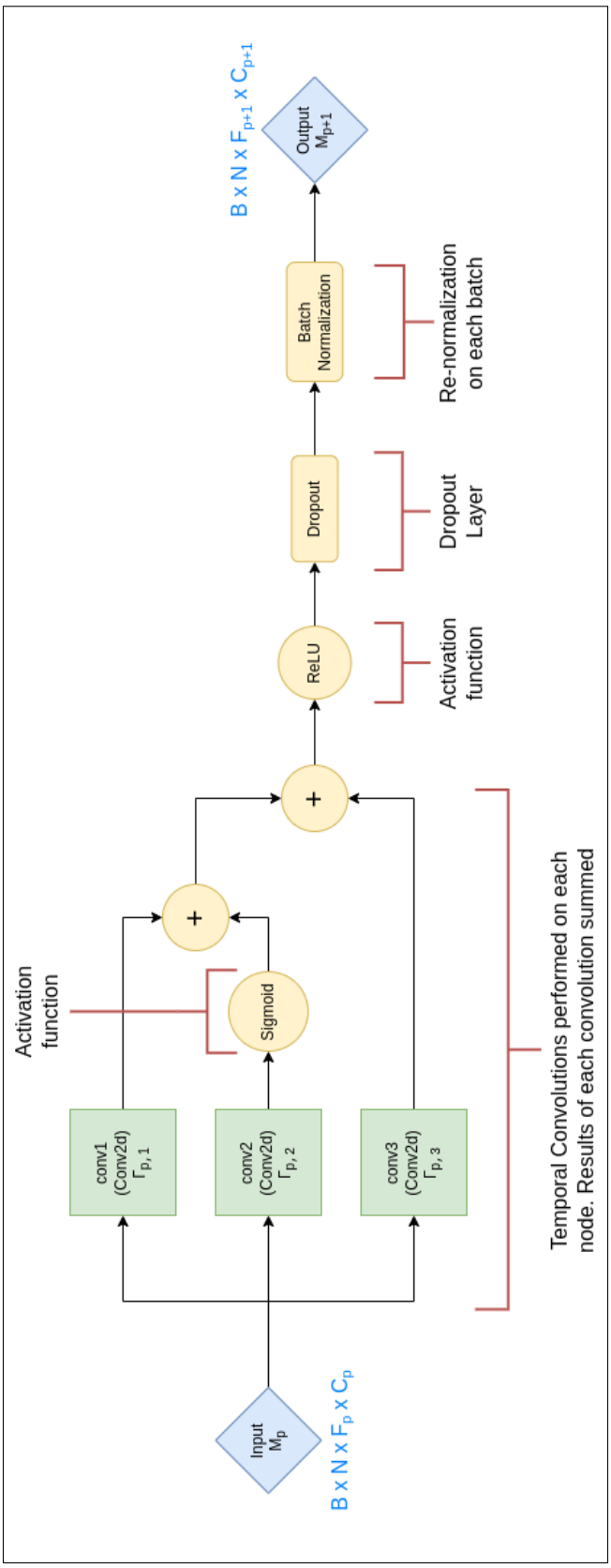


Figure 3.9: Structure of my Time Block

`conv1`, `conv2`, `conv3`

These perform 1D convolution on each node's feature vector dimensions. I use the `Conv2D` function to enact this but set one of the dimensions to 1. Kernel $\Gamma_{p,x}$ (from `convx`) is a one dimensional array of length K with values $[\gamma_{x,0}, ..., \gamma_{x,K-1}]$. The input feature vector is denoted as $[f_{p,0}, ..., f_{p,F_p}]$ and the returned vector as $[f_{p+1,0}, ..., f_{p+1,F_{p+1}}]$.

$$f_{p+1,j} = \sum_{i=0}^{K} f_{p,j+i}\gamma_{x,i}, \quad 0 \leq j \leq F_p - K \tag{3.14}$$

This equation is represented in equation 3.13 as $\Gamma_{p,x} *_{1D} M_p$. The output feature vector length F_{p+1} will equal to $F_p - K + 1 = F_p - 2$ since, at $i = K$ and $j = F_p - K$, we will have reached the end of my input vector $f_{p,F_p-K+K} = f_{p,F_p}$.

We use 3 convolutional modules to allow for modelling of 3 traffic patterns; immediate, daily, and weekly.

Sigmoid, ReLU

Sigmoid and ReLU are activation functions which perform the same role as the ReLU in my STGCN Block. The sigmoid activation function is described by the equation

$$\boldsymbol{\sigma}(x) = \frac{1}{1 + e^{-x}} \tag{3.15}$$

Dropout, Batch Normalization

Dropout and batch normalization are done in the same way as in the STGCN Block, with the same dropout rate of 0.3.

3.3.5 Model Size

The size of my model, with respect to its initialization values, is shown in Table 3.2. Since each batch B is run through the same parameters, this dimension does not affect the model parameters. My size is defined entirely by the number of channels in my input C_O, the number of channels created by my spatial convolution C_S, the number of channels created by my temporal convolutions C_{p+1}, the number of nodes

Parameter Set Name	**Number of Parameters**
block1.Theta1	$N * C_{p+1} * C_S$
block1.temporal1.conv1.weight	$C_I * C_{p+1} * K$
block1.temporal1.conv1.bias	C_{p+1}
block1.temporal1.conv2.weight	$C_I * C_{p+1} * K$
block1.temporal1.conv2.bias	C_{p+1}
block1.temporal1.conv3.weight	$C_I * C_{p+1} * K$
block1.temporal1.conv3.bias	C_{p+1}
block1.temporal2.conv1.weight	$C_S * C_{p+1} * K$
block1.temporal2.conv1.bias	C_{p+1}
block1.temporal2.conv2.weight	$C_S * C_{p+1} * K$
block1.temporal2.conv2.bias	C_{p+1}
block1.temporal2.conv3.weight	$C_S * C_{p+1} * K$
block1.temporal2.conv3.bias	C_{p+1}
block1.batch_norm.weight	N
block1.batch_norm.bias	N
block2.Theta1	$N * C_{p+1} * C_S$
block2.temporal1.conv1.weight	$C_{p+1} * C_{p+1} * K$
block2.temporal1.conv1.bias	C_{p+1}
block2.temporal1.conv2.weight	$C_{p+1} * C_{p+1} * K$
block2.temporal1.conv2.bias	C_{p+1}
block2.temporal1.conv3.weight	$C_{p+1} * C_{p+1} * K$
block2.temporal1.conv3.bias	C_{p+1}
block2.temporal2.conv1.weight	$C_S * C_{p+1} * K$
block2.temporal2.conv1.bias	C_{p+1}
block2.temporal2.conv2.weight	$C_S * C_{p+1} * K$
block2.temporal2.conv2.bias	C_{p+1}
block2.temporal2.conv3.weight	$C_S * C_{p+1} * K$
block2.temporal2.conv3.bias	C_{p+1}
block2.batch_norm.weight	N
block2.batch_norm.bias	N
fully.weight	$C_{p+1} * C_O * (F_I - 4 * (K - 1))$
fully.bias	C_O

Table 3.2: Number of Parameters in my Model w.r.t. initialization values

N, the size of the input feature vector F_I, and the size of my kernel K.

When we add these together we get the equation below, which can help us to understand how my equation scales.

$$C_{p+1}\left(2C_S N + 3K\left(C_I + C_{p+1} + 2C_I\right) + 12 + C_O * (F_I - 4 * (K-1))\right) + 4N + C_O \tag{3.16}$$

C_{p+1} clearly has a large influence on the overall size as this is the channel size we see most in my model. However, since K is so small, changes to K would have a huge

impact on the overall size.

3.4 Model Training

I use the Adam optimization algorithm for training my model parameters. The parameters I use for this algorithm are $\alpha = 0.001$ (also called the learning rate), $\beta_1 = 0.9$, and $\beta_2 = 0.99$. These parameters determine how the model searches through the parameter space. I use a minibatch size of 32, which means that the algorithm takes a sample of 32 random instances from my training set and uses the loss function on these to update my parameters. I continue to take random batches until all of my training instances have been used, constituting on epoch.

I use Mean Squared error (MSE) as my loss function (except when outputting a distribution as seen in Section 3.6). The loss gradient is found using back propagation which is inbuilt in PyTorch.

3.5 Performance Metrics

I will use 3 metrics for comparing my models. The equations for these are shown below with x_i as the true value for instance i, $\hat{x}_i$ as my predicted value, and n as the number of instances being tested.

1. Mean Absolute Error (MAE): This is the error used in training my SVR model.

$$\text{MAE} = \frac{\sum_{i=1}^{n} |x_i - \hat{x}_i|}{n} \tag{3.17}$$

2. Mean Absolute Percentage Error (MAPE): Although this is given as a percentage it is possible to exceed 100% in cases where the forecast is much greater than the true value. In my case it is possible for noise to cause the true value to be much lower than forecasted. In cases where the true value is zero, the absolute difference is divided by ϵ (an arbitrarily small positive number) to avoid indeterminate results. In the Scikit-learn library, (4) $\epsilon = 2^{-52} \approx 2.2 \times 10^{-16}$, the lowest 64-bit floating point value possible. This can cause MAPE results

to be very high. I have included MAPE in my results, but show in Section 4.7 why it is unreliable.

$$\text{MAPE} = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{x_i - \hat{x_i}}{\max(x_i, \epsilon)}\right| \tag{3.18}$$

3. Root-Mean-Square Error (RMSE): Since MSE is used as my loss function, we would say that the best model is the one that performs best in terms of RMSE. If I valued one of the other metrics more, I could retrain the model with that as my loss function.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x_i})^2} \tag{3.19}$$

3.6 Estimate Distribution

In order to provide a better understanding of my predictions, it is possible to create a model that outputs a prediction as well as a confidence interval. To do this, I can simply change my model to output two values for every prediction (making $C_O = 2$ instead of 1 as before). I can train these to represent mean μ and standard deviation σ by using the Negative Log Probability (NLP) (32) as my loss function, instead of MSE as before.

For each sample and node I will output a $\mu_{\hat{x}}$ and $\sigma_{\hat{x}}$ and use the PyTorch `distributions.Normal` function to create a normal distribution with probability density function (p.d.f.):

$$p(x) = \frac{1}{\sigma_{\hat{x}}\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{x-\mu_{\hat{x}}}{\sigma_{\hat{x}}}\right)^2} \tag{3.20}$$

For one variable *likelihood* that my true value x belongs to my normal distribution is equivalent to the p.d.f. at x. For one variable this is equivalent to the probability density function. Log probabilities are more practical for computation as they are on a logarithmic scale rather than between 0 and 1. This means that small differences can be seen more easily. I take the negative of this as the convention is to *minimize*

loss functions in PyTorch.

$$-\ln p(x) = \frac{1}{2}\left(\frac{x-\mu_{\hat{x}}}{\sigma_{\hat{x}}}\right)^2 + \ln\left(\sigma_{\hat{x}}\sqrt{2\pi}\right) \tag{3.21}$$

The average log probabilities of all my predictions is done using the PyTorch function `log_prob` and the negative of this is taken. B and N are the batch size and number of nodes, as described in Section 3.3.

$$\text{NLP} = -\frac{1}{B*N}\sum_{i=0}^{B-1}\left(\sum_{n=0}^{N-1}\ln p(x)\right) \tag{3.22}$$

With two outputs I would expect this to take longer to train. However, once it does, I should expect $\mu_{\hat{x}}$ to be as good a predictor for x as the $\hat{x}$ predicted by my initial model. If the standard deviation accurately represents the uncertainty accurately, I would expect to see approximately 68.27% of the true values fall within one standard deviation of my mean.

3.7 Baseline Model

I chose to use a Support Vector Regressor (SVR) model as my baseline. SVRs has been used to forecast flow in (5), though my baseline SVR is simpler and does not use a genetic algorithm. SVRs are a class of machine learning model that perform the 'kernel trick' to map data into a higher dimensional feature space. Once the kernel is properly trained, we can perform a regression in this higher dimensional space.

We create an SVR model for each node and train them individually. This means that they can only represent the temporal, and not the spatial, relationships. My SVR model uses Mean Absolute Error (MAE) as its loss function and runs for 1000 iterations. Each model creates 25 coefficients (one for each value in the feature vector), 10 hyperspace parameters (used for creating the higher dimensional features) and 1 intercept (for scaling the results).

This gives 36 parameters per node in my junction set. This is much fewer than the number of parameters in the models I create in this report. However, the number

of parameters increase much faster in the SVR when we increase the number of nodes (for example the PeMSD7 data set model contains 207 nodes meaning $207 * 36$ parameters) or the size of the feature vector. The DC-STGCN model size, on the other hand, is largely determined by the internal channels which remain constant. We can see from Equation 3.16 that increasing N or F_I would have a relatively small effect.

Chapter 4

Experiments & Results

I first describe the time periods and junction sets used in training and testing my models. Next, I show the results of models that I trained on each of my 3 junction sets. I then look at the results of my estimate distribution model, followed by the results for the single channel model. Lastly, I discuss the results from each of my experiments, and their implications.

4.1 Time Periods Used

4.1.1 Basic Train, Validation, Test Data

The initial dataset for my training, validation and testing comes from April 1st 2018 to June 21st 2018 (82 days). Due to the way my feature vector is created, the first week of this cannot be tested. This leaves us with 75 days of data; 52 weekdays, 21 weekend days and 2 bank holidays (7th May, 4th June). With 3 minutes per interval, this is 36,000 intervals that I can use. As this period includes Easter school holidays (March 24th to April 8th) and the beginning of summer holidays (in June, varying between different schools and universities) I would expect this to create a robust model.

We use a 50:40:10 split which can be completely shuffled as the feature vectors have already been created. However, for convenience in analysis, I wanted the test data to be sequential and consisting of a whole number of days. For this I chose the test data to be from between May 14th and May 22nd (11.11%) as this is roughly

halfway through the data. I will refer to this as the original (OG) test period. The remaining data is shuffled and distributed between training and validation with a 5:4 split.

4.1.2 Further Testing Periods

I chose 4 more testing weeks for analysing the robustness of the model to new data. All of these periods are from Monday to Sunday.

- New Year (NY) - Monday January 8th to Sunday January 14th: This is the week that schools reopened from Christmas holidays. In the previous week (January 1st to January 7th) schools were still closed and people were coming back to work after New Year's, so the feature vector components from the previous week should be unreliable for predictions.

- January into February (JF) - Monday February 5th to Sunday February 11th: Both this week and the week previous (January 29th to February 4th) were in the middle of school term and contained no other aberrations.

- Mid-Term Break (MT) - Monday February 12th to Sunday February 18th: This week sees primary and secondary schools close with many people taking holidays, making the traffic flows unpredictable.

- Summer Holidays (SH) - Sunday June 24th to Saturday June 30th: This is the final test period used. It is the start of summer, with all schools and universities closed at this point.

4.2 Junction Sets

4.2.1 Naming Convention

Seeing as there are several terms with seemingly similar meanings, I will clarify how I define each. These terms are capitalized when referring to a specific example.

- Junction: A junction is a real world traffic junction that contains several sensors, and have names like 48 and 186.

- Sensor: A sensor is the device which measures the flow and density on a particular entry route to a junction. These have names such as 48iii or 186i.

- Subsystem: Sensors are grouped by subsystem in the SCATS files. I select the subsystem containing the sensors I wish to analyse in order to create my text files. Those used here have values ranging from 3 to 85.

- Node: Each sensor corresponds to a node in my graph structure. Once the data from a sensor is read in, I refer to it as a node, and the data as the node values. These have indexes between 0 and the number of vertices in my graph.

- Set: A set is a group of nodes that will be analysed in a model. These nodes in turn correspond to a sensor which is inside a real world junction. The names of my sets are **1**, **1+**, and **2**, and are written in bold.

4.2.2 Junction Set 1

Junction Set **1** was chosen to spread out my nodes as much as possible. These were the most distant nodes available in my SV files. Having the nodes far apart increases the ability to utilize our spatial data. It is more likely that we will be able to see traffic patterns coming from other nodes ahead of time when it takes longer to for said patterns to travel between them. If two nodes are too close they are considered contemporaneous, meaning that patterns will arrive to them at the same time or within the same interval.

Table 4.1 shows the details of the 4 junctions I chose for this set. These junctions are shown on a map in Figure 4.1. Figure 4.2 shows the junctions' internal structure, with the node indexes labelled along with the routes that can be taken from each node. Figure 4.3 shows the complete distance matrix for Set **1**. The maximum distance between two nodes is 3.24 kilometres.

We can use autocorrelation to gauge how well a node can be predicted from previous time intervals. Nodes with peaks of greater amplitude will be more easily predicted through temporal relations alone (as is the case with my SVR model). Seeing as I am using intervals in my feature vector that correspond to approximately integer numbers of days prior to my prediction we only need to observe the positive

Junction Number	Subsystem Number	Number of Nodes	Street Names (Number of Lanes)
48	28	3	Lord Edward St. (2), Dame St. (1), Parliament St. (2)
145	3	3	Pearse St. (2), Sandwith St. Upper (1), Sandwith St. Lower (3)
152	23	3	Usher's Quay (3), Ellis Quay (2), Queen's St. (3)
354	10	2	Amiens St. (3), Memorial Rd. (3)

Table 4.1: Junctions in Set **1**

Figure 4.1: Map of Set **1**

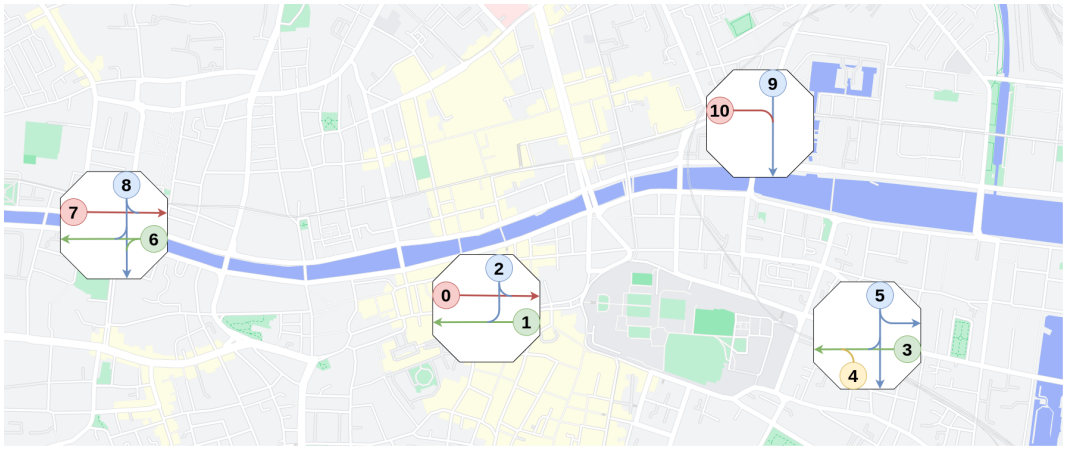


Figure 4.2: Map of Set **1** with Node labels and internal junction structure

amplitude.

Figure 4.4 shows the autocorrelation plots from the nodes in Set **1**. In Set **1** the standout nodes are Node 6 with, very high autocorrelation, and Node 1, with very low autocorrelation. The other nodes have similar plots.

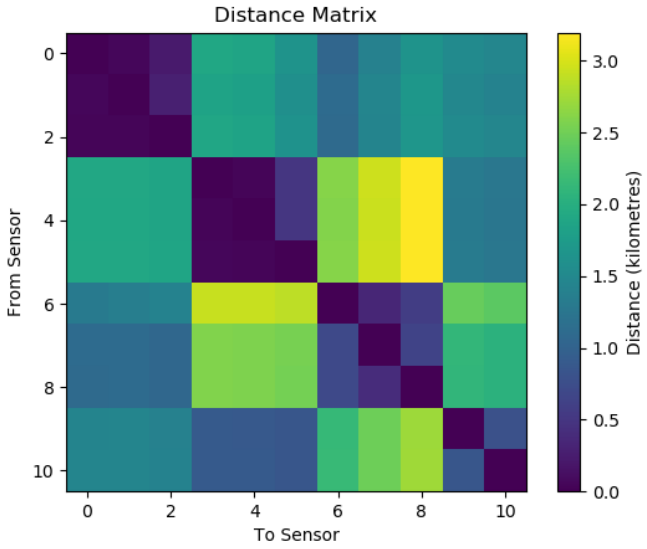


Figure 4.3: Complete Distance Matrix for Set **1**

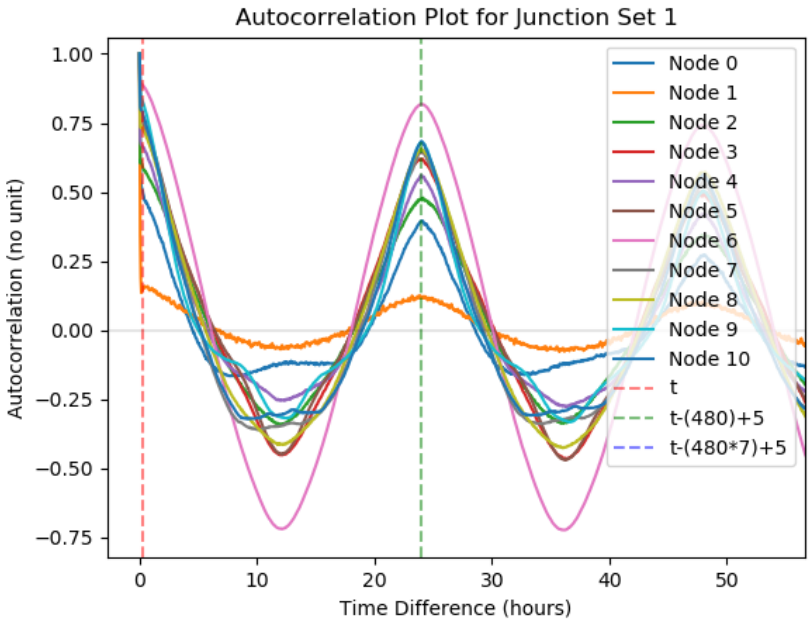


Figure 4.4: Autocorrelation Plot for Set **1**

4.2.3 Junction Set 1+

Junction Set **1+** is an expansion of Set **1**, with one extra junction (Junction 196). This expansion will be compared with the results of Set **1** to see how my performance on the same nodes changes with a different network.

Table 4.2 details the junctions, with Figure 4.5 showing their location on a map. Figure 4.6 shows the inner structure and node indexes of my set. Figure 4.3 shows

the complete distance matrix for Set **1+**. The maximum distance is the same here as in Set **1**.

Junction Number	**Subsystem Number**	**Number of Nodes**	**tStreet Names (Number of Lanes)**
48	28	3	Lord Edward St. (2), Dame St. (1), Parliament St. (2)
145	3	3	Pearse St. (2), Sandwith St. Upper (1), Sandwith St. Lower (3)
152	23	3	Usher's Quay (3), Ellis Quay (2), Queen's St. (3)
354	10	2	Amiens St. (3), Memorial Rd. (3)
196	85	3	Westmoreland St. (3), O' Connell's Street (4), Burgh Quay (4)

Table 4.2: Junctions in Set **1+**

Figure 4.5: Map of Set **1+**

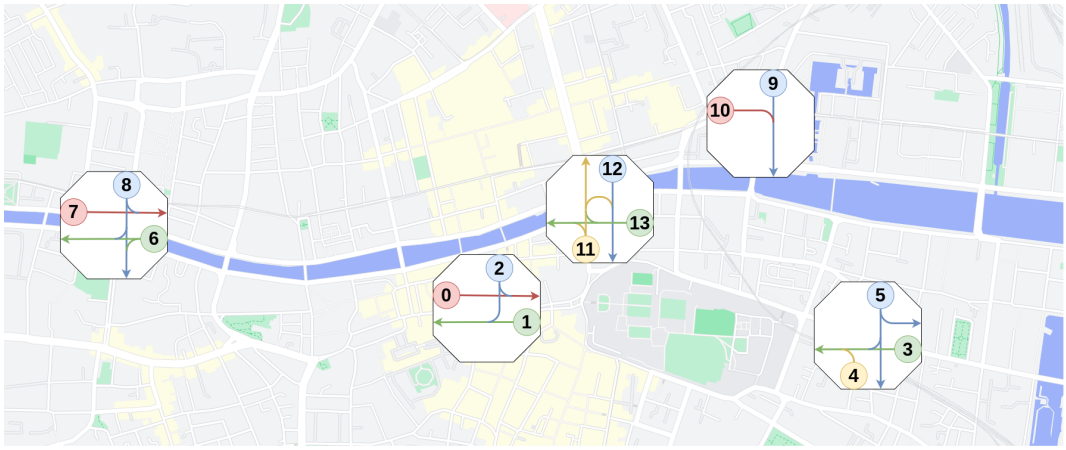


Figure 4.6: Map of Set **1+** with Node labels and internal junction structure

Figure 4.8 shows the autocorrelation plot for Set **1+**. The new nodes fall within the same range as most of my Set **1** nodes, with Nodes 1 and 6 having the lowest

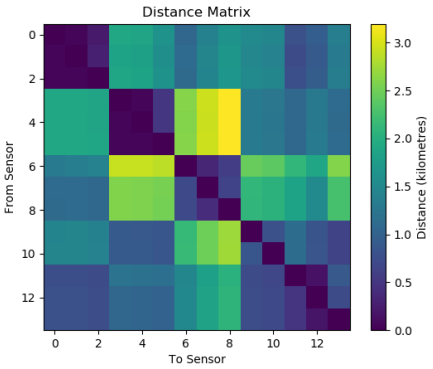


Figure 4.7: Complete Distance Matrix for Set **1+**

and highest values, respectively.

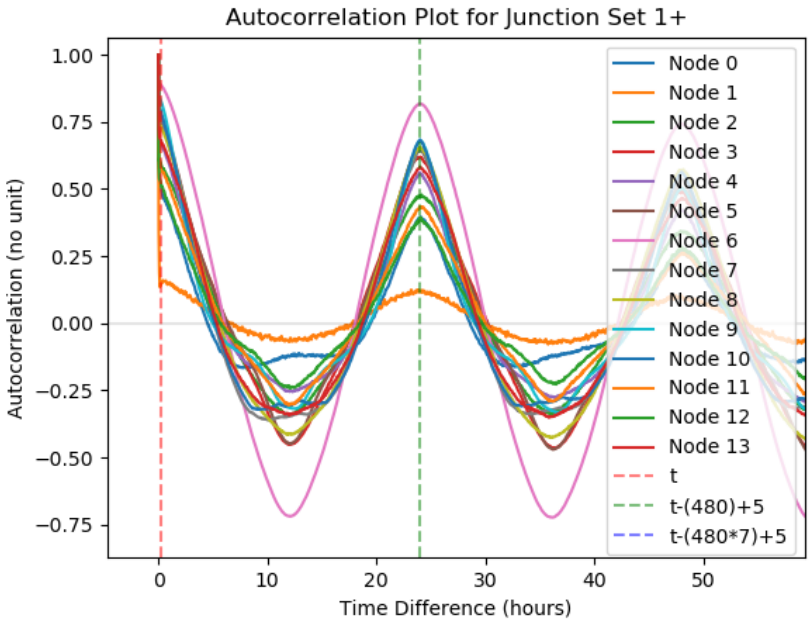


Figure 4.8: Autocorrelation Plot for Set **1+**

4.2.4 Junction Set 2

Junction Set **2** contains much closer junctions, which could be considered contemporaneous. Set **2** shares one junction (2 nodes) with Set **1** and two junctions (5 nodes) with Set **1+** This will be a useful comparison to see what difference contemporaneous nodes makes in my spatial convolution. I chose junctions situated between Trinity College Dublin and the River Liffey.

Table 4.3 gives the details of the junctions in Set **2**. Figures 4.9 and 4.10 show the junction locations and internal structures respectively. Figure 4.11 shows the complete distance matrix of Set **2**. The distances here are a lot smaller, with the greatest being 1.28 kilometres.

Junction Number	**Subsystem Number**	**Number of Nodes**	**Street Names (Number of Lanes)**
183	6	2	Tara St. (4), George's Quay (3)
196	85	3	Westmoreland St. (3), O' Connell's Street (4), Burgh Quay (4)
288	58	2	Pearse Street (4), Shaw St. (2)
354	10	2	Amiens St. (3), Memorial Rd. (3)

Table 4.3: Junctions in the Set **2**

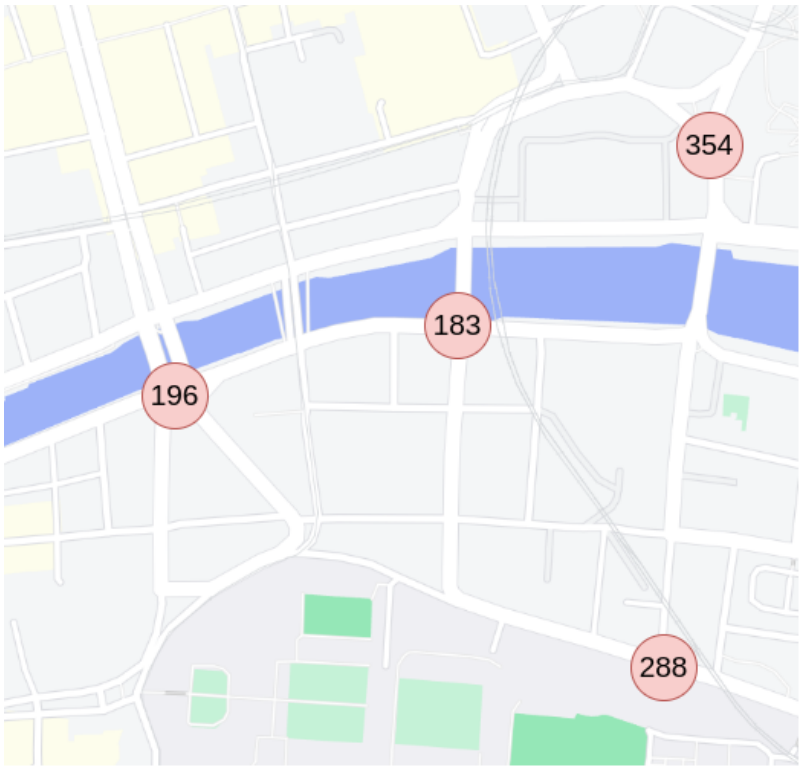


Figure 4.9: Map of Set **2**

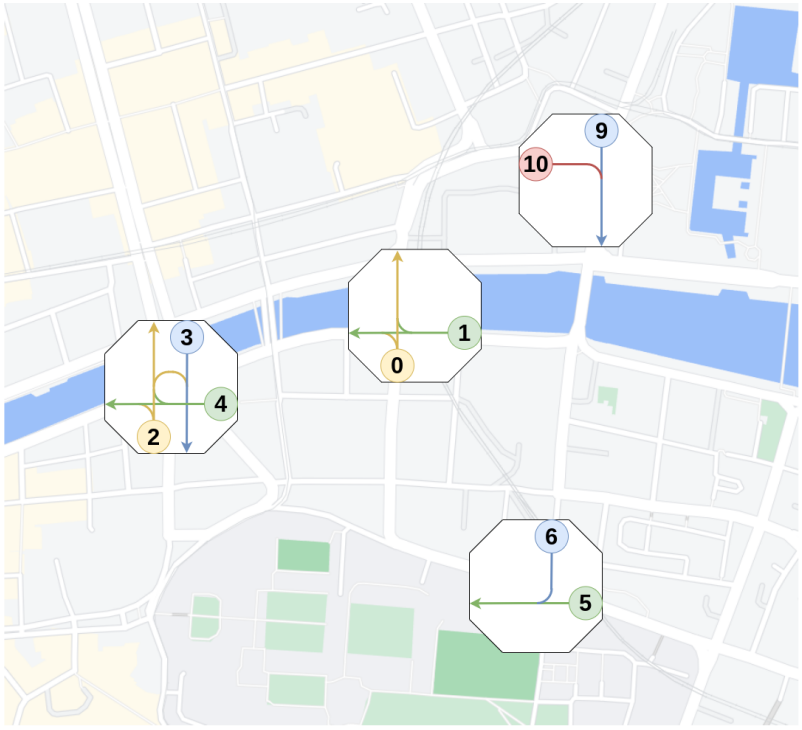


Figure 4.10: Map of Set **2** with Node labels and internal junction structure

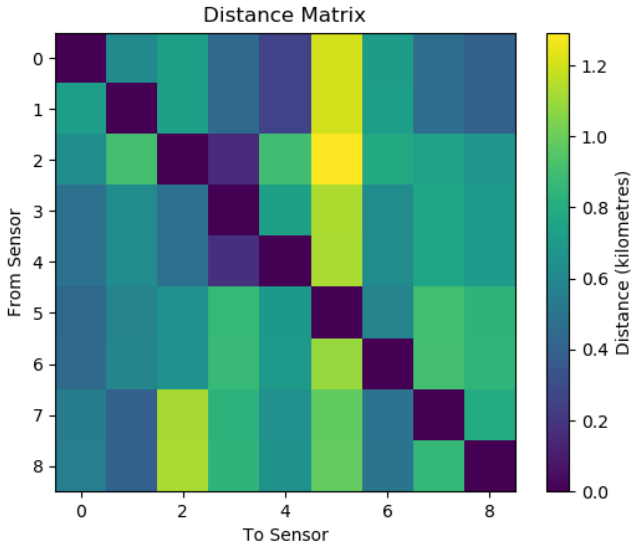


Figure 4.11: Complete Distance Matrix for Set **2**

Figure 4.12 shows the autocorrelation plots for the nodes in Set **2**. Nodes 0 and 5 have the highest autocorrelations with Node 8 peaking slightly lower. There are no nodes with particularly low autocorrelation peaks in Set **2**.

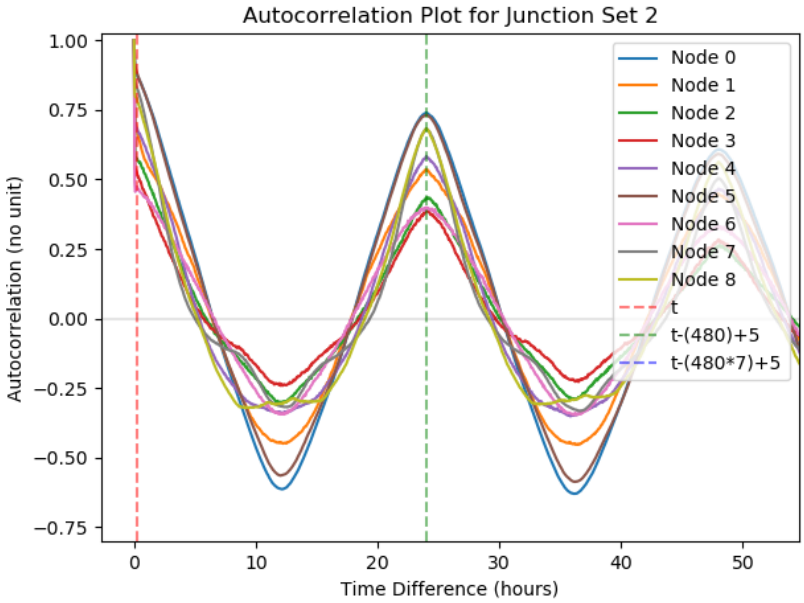


Figure 4.12: Autocorrelation Plot for Set **1+**

4.2.5 Junction Set Comparison

Table 4.4 shows the node indices for each sensor I am using across my different junctions. I have also included the Average Flow and Standard Deviation of Flow in my test period to help create an understanding of the behaviour of these nodes. This will be useful when comparing model performance on a specific node.

	1	**1+**	**2**	**Avg Flow**	**Std Dev**
48, i	0	0	-	586.2	302.1
48, ii	1	1	-	444.0	325.3
48, iii	2	2	-	784.5	354.2
145, i	3	3	-	729.1	384.9
145, ii	4	4	-	274.2	166.4
145, iii	5	5	-	1342.0	800.1
152, i	6	6	-	2369.3	1143.5
152, ii	7	7	-	1079.7	502.3
152, iii	8	8	-	1723.9	797.4
183, i	-	-	0	1521.9	768.3
183, ii	-	-	1	1282.4	550.6
196, i	-	11	2	1147.2	443.3
196, ii	-	12	3	1129.2	514.8
196, iii	-	13	4	1871.9	694.4
288, i	-	-	5	1354.1	725.6
288, ii	-	-	6	511.6	483.9
354, i	9	9	7	1144.9	583.3
354, ii	10	10	8	1351.9	709.6

Table 4.4: Junctions, Node Indices, and Flow Characteristics

4.3 Experiment Details

Appendix A contains details on each model used in this section. All models have been trained for 1,000 epochs unless otherwise specified.

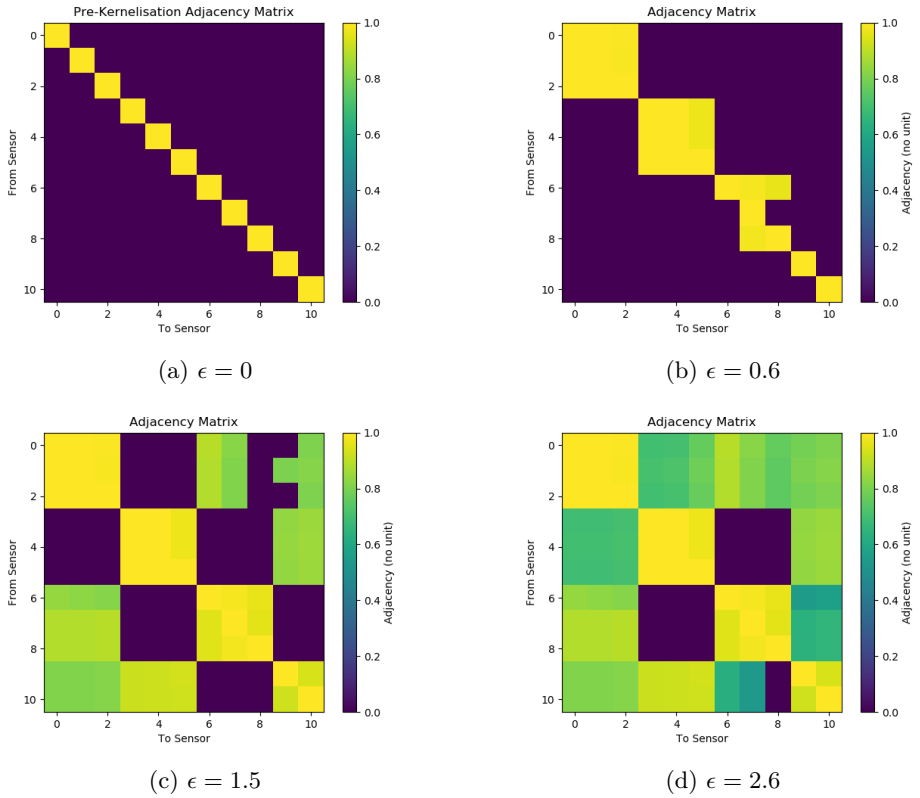
4.4 Set 1 Experiments

4.4.1 Changing Kernelization Values

My first experiment was to compare the performance of models that use different parameters in the kernelization of their distance matrix. Figure 4.13 shows the effect of changing my ϵ parameter, on a distance matrix with no pruning. All models on Set **1**, using 2 input channels and no estimate distribution, contain 60,397 parameters. for reference, a model of this size corresponds to 253.8 kilobytes of storage space.

Table 4.5 shows the performance of models with different ϵ values (holding δ^2 constant at 10). We can see that the model which used $\epsilon = 1.5$ had the best performance in 2 of the performance criteria. I will use this value moving forward in the next experiments.

An epsilon value of means that only each node's connection to itself is maintained



(a) $\epsilon = 0$

(b) $\epsilon = 0.6$

(c) $\epsilon = 1.5$

(d) $\epsilon = 2.6$

Figure 4.13: Adjacency Matrices created using different ϵ values

(as these are the only places in the distance matrix of value 0), giving a diagonal matrix. In this case my spatial convolution would have no effect. This is included as a sanity check to ensure that the prediction is improved upon by the spatial modelling, which is indeed the case. The diagonal matrix achives a lower MAPE score than the other models.

Epsilon Value	**MAE**	**MAPE**	**RMSE**
0	239.87	**5.037E+016**	312.28
0.6	244.14	6.891E+016	313.19
1.5	**235.21**	5.216E+016	**306.89**
2.6	241.65	5.313E+016	313.51

Table 4.5: Effect of Epsilon Value on Set **1** performance

4.4.2 Adjacency Matrix Pruning

Our next experiment is to evaluate the model performance when different levels of pruning are applied to my adjacency matrix. I look at three options:

- Complete: In this case I do not alter the adjacency matrix manually. This is

the same as the model that performed best in the last section.

- Same Junction Pruning: Here I set the adjacency between two different nodes of the same junction as 0. These are the most unlikely routes that a driver might take.

- Unreasonable Route Pruning: This involves creating a list of reasonable routes that a driver might take and only including the adjacency values of these routes (along with the values of 1 in the diagonal). Figure 4.14 shows the reasonable routes that I chose for Set **1**. These were decided upon by looking at the directions that a driver can take for each node and seeing if there was a reasonable route to take in order to reach another of my nodes. I would expect this method to best model the spatial dependencies.

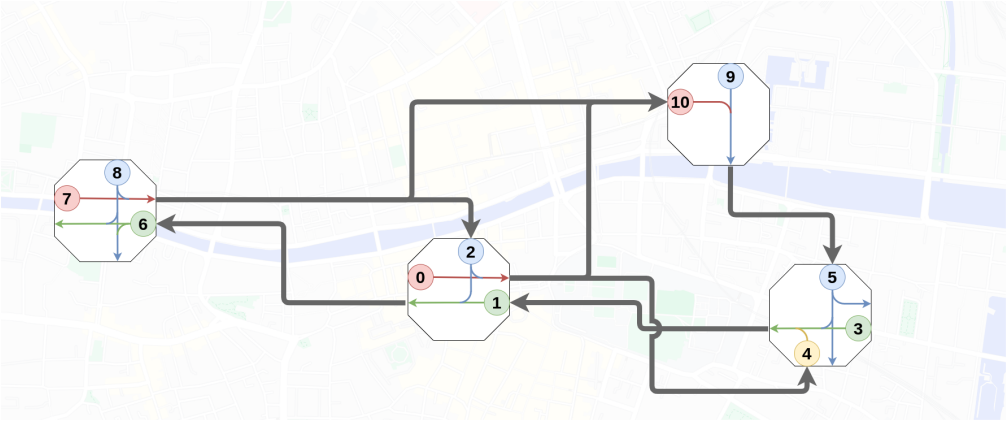


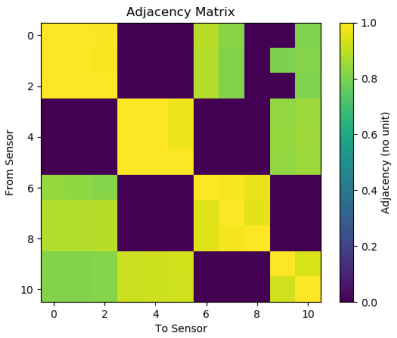
Figure 4.14: Reasonable Routes chosen for Set **1**

Figure 4.15 shows the adjacency matrices corresponding to each of these pruning options.

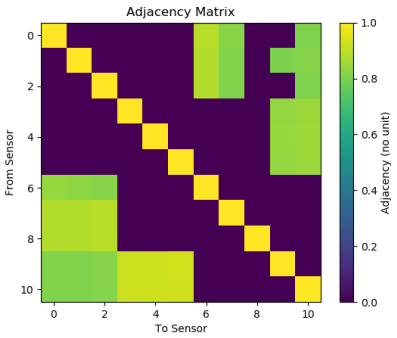
Table 4.6 shows the performance of models that use each of these pruning options. The Same Junction Pruning performs in terms of MAE and RMSE, while the Realistic Route Pruning outperforms it on MAPE.

Adjacency Matrix Type	**MAE**	**MAPE**	**RMSE**
Complete	235.21	**5.216E+016**	306.89
Same Junction Pruning	**228.42**	5.492E+016	**298.63**
Unreasonable Route Pruning	235.57	5.743E+016	304.46

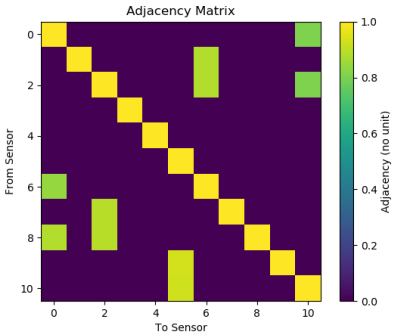
Table 4.6: Effect of Adjacency Matrix Pruning on Set **1** Performance



(a) Un-Pruned



(b) Same Junction Pruning



(c) Unreasonable Route Pruning

Figure 4.15: Adjacency Matrix Pruning for Set **1**

4.4.3 Performance against my Baseline Model on OG

Using $\epsilon = 1.5$ and Same Junction Pruning (as these performed best in my previous experiments) I will compare my STGCN model with my baseline SVR. Although training appears to plateau around 1000 epochs, I trained for 4000 epochs to see if there was improvement. The results are compared to my SVR results in Table 4.7. We can see that there is a marginal improvement from 1000 to 4000 epochs on every metric. Both of my STGCN models outperform the SVR on MAE and RMSE, but my SVR models give a better MAPE score. I will use the 4000 epoch model in the comparison across nodes and days.

Model	**MAE**	**MAPE**	**RMSE**
STGCN (1000 epochs)	228.42	5.492E+016	298.63
STGCN (4000 epochs)	**227.30**	5.123E+016	**298.38**
SVR	231.76	**4.298E+016**	306.76

Table 4.7: STGCN performance against my baseline on Set **1**

Table 4.8 shows how the performance of my STGCN and SVR models compare across the different nodes in my set. The STGCN performs better in terms of MAE

and RMSE in most of my nodes. The nodes it does not beat the SVR on, by these metrics, are Nodes 1, 4, and 6. I will look at these nodes in Section 4.11. The SVR outperforms my STGCN in terms of MAPE on all nodes apart from Nodes 2 and 4.

	STGCN			**SVR**		
Node	**MAE**	**MAPE**	**RMSE**	**MAE**	**MAPE**	**RMSE**
0	**162.75**	8.308E+15	**213.40**	165.45	**7.798E+15**	222.43
1	218.84	1.606E+17	283.83	**217.91**	**1.568E+17**	**282.50**
2	**214.02**	**4.982E+16**	**273.03**	217.23	5.033E+16	277.47
3	**169.03**	8.975E+14	**250.18**	174.32	**8.016E+14**	253.36
4	91.50	**1.477E+16**	120.54	**80.85**	2.040E+16	**110.20**
5	**288.50**	1.090E+17	**372.14**	301.39	**6.246E+16**	396.85
6	342.28	1.346E+15	436.10	**339.28**	**1.172E+15**	**435.89**
7	**192.72**	3.486E+15	**254.66**	195.33	**3.317E+15**	261.43
8	**303.35**	4.411E+16	**388.79**	316.17	**3.812E+16**	410.49
9	**249.49**	**1.591E+16**	**346.50**	260.88	**1.592E+16**	362.13
10	**267.83**	1.554E+17	**343.11**	280.59	**1.157E+17**	361.71
Average	**227.30**	5.124E+16	**298.39**	231.77	**4.299E+16**	306.77

Table 4.8: STGCN performance against my baseline on Nodes in Set **1**

Table 4.9 shows how my STGCN and baseline perform on each day of the OG period. SVR performs better in MAPE and RMSE on almost all junctions. The only days when the STGCN performed worse than the SVR in terms of MAE were Saturday and Sunday. I will discuss the RMSE score in Section 4.11.

	STGCN			**SVR**		
Day	**MAE**	**MAPE**	**RMSE**	**MAE**	**MAPE**	**RMSE**
0 (Mon)	**231.38**	6.252E+16	**316.86**	260.84	**5.413E+16**	338.41
1 (Tues)	**220.72**	4.206E+16	306.35	223.10	**3.099E+16**	**298.53**
2 (Wed)	**232.01**	5.817E+16	321.65	233.19	**4.589E+16**	**310.72**
3 (Thurs)	**234.92**	4.834E+16	316.83	237.19	**3.844E+16**	**310.40**
4 (Fri)	**239.78**	4.848E+16	323.43	243.08	**4.246E+16**	**315.39**
5 (Sat)	230.39	4.194E+16	310.80	**228.08**	**3.580E+16**	**297.45**
6 (Sun)	219.14	5.046E+16	293.12	**214.84**	**4.609E+16**	**275.74**
7 (Mon)	**218.53**	5.918E+16	303.12	225.80	**4.924E+16**	**302.88**
8 (Tues)	**218.83**	5.000E+16	300.62	223.21	**3.624E+16**	**298.47**

Table 4.9: STGCN performance against my baseline on Days in OG for Set **1**

Performance on Test Sets

The results for my additional test sets are shown in Table 4.10. The STGCN does best in terms of MAE and RMSE on OG and SH only. For periods NY, JF, and

MT, the SVR shows the best performance in terms of MAE and MAPE. Figure 4.16 shows a breakdown of STGCN performance (in MAE) by nodes for my different test nodes. We can see the highest variability in performance is at Node 1 by quite some margin.

	STGCN			SVR		
Test Set	**MAE**	**MAPE**	**RMSE**	**MAE**	**MAPE**	**RMSE**
OG	**227.300**	5.124E+16	**298.389**	231.905	**4.314E+16**	306.816
NY	224.048	6.809E+16	293.301	**219.308**	**5.939E+16**	**292.408**
JF	220.800	5.898E+16	**287.691**	**218.068**	**4.850E+16**	288.729
MT	221.545	5.992E+16	**287.670**	**220.643**	**4.939E+16**	290.681
SH	**215.583**	1.936E+05	**282.145**	216.523	**1.290E+05**	286.717

Table 4.10: Performance on Test Sets for Set **1**

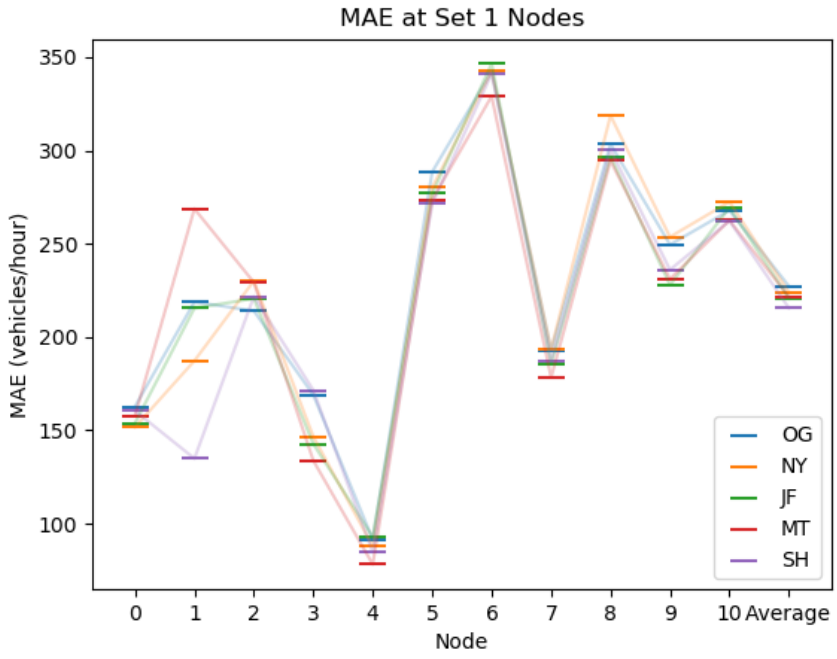


Figure 4.16: Performance on Test periods by Node for Set **1**

4.5 Set 1+ Experiments

4.5.1 Adjacency Matrix

In testing on my Set **1+** junctions I used the hyperparameters that were found to be best on Set **1**; $\epsilon = 1.5$ and Same Junction Pruning. The pruned adjacency matrix for Set **1+** is shown in Figure 4.17. The additional junctions not in Set **1** are Nodes

11, 12, and 13, corresponding to the last three rows and columns of the adjacency matrix.

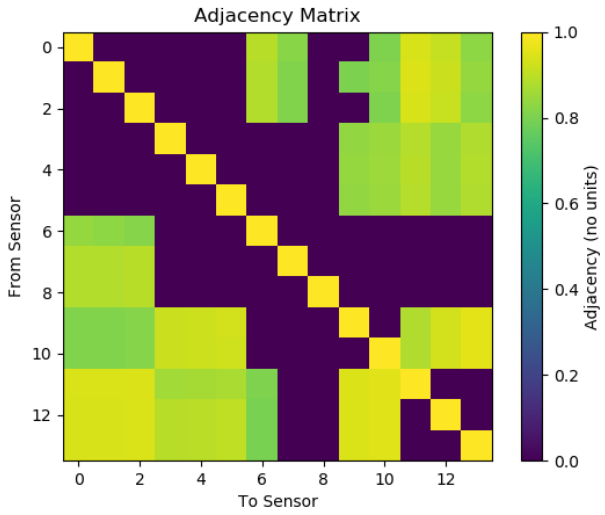


Figure 4.17: Set **1+** adjacency matrix with same junction pruning

4.5.2 Results on OG test period

Table 4.11 shows the comparison of my model with the baseline on Set **1+**, and is similar to that of Set **1**. The STGCN performs best in terms of MAE and RMSE, while the SVR achieves a lower MAPE score.

Model	**MAE**	**MAPE**	**RMSE**
STGCN (4000 epochs)	**243.34**	4.228E+16	**317.52**
SVR	245.25	**3.728E+16**	323.47

Table 4.11: STGCN performance against my baseline on Set **1+**

The performance is broken down by node in Table 4.12. These results are again similar to those for Set **1**, with STGCN performing best on most junctions for MAE and RMSE, and SVR performing best on most in terms of MAPE. However, the nodes on which SVR achieves a lower MAE score are different to those in Set **1**. In Set **1+**, these are Nodes 0, 2, 6, 7, and 12.

4.5.3 Performance on Test Periods

Table 4.13 shows the performance of my Set **1+** STGCN against my SVR baseline. The SVR model achieves the best MAE scores for 4 of my test periods, while the

	STGCN			**SVR**		
Node	**MAE**	**MAPE**	**RMSE**	**MAE**	**MAPE**	**RMSE**
0	172.62	9.580E+15	221.07	**165.51**	**8.843E+15**	**220.23**
1	**216.39**	1.599E+17	**282.13**	217.77	**1.568E+17**	282.63
2	219.72	5.519E+16	280.02	**216.88**	**5.290E+16**	**276.64**
3	**169.86**	8.977E+14	**249.35**	174.56	**7.814E+14**	253.7
4	**79.87**	**2.010E+16**	**107.17**	80.4	2.183E+16	109.36
5	**292.55**	1.065E+17	**376.83**	301.17	**8.000E+16**	394.83
6	343.25	1.305E+15	**436.17**	**342.24**	**1.164E+15**	439.43
7	196.56	4.415E+15	**254.60**	**194.97**	**3.361E+15**	260.94
8	**304.90**	4.309E+16	**391.45**	315.27	**3.723E+16**	409.27
9	**256.50**	1.625E+16	**353.39**	260.77	**1.603E+16**	362.06
10	**272.45**	1.399E+17	**348.15**	279.62	**1.099E+17**	360.83
11	**258.02**	3.337E+15	**339.49**	**258.02**	**2.802E+15**	342.22
12	312.67	8.423E+15	**403.41**	**308.47**	**8.292E+15**	405.4
13	**311.41**	2.295E+16	**402.10**	317.84	**2.202E+16**	411
Average	**243.34**	4.228E+16	**317.52**	245.25	**3.728E+16**	323.47

Table 4.12: STGCN performance against my baseline on Set **1+** Nodes

	STGCN			**SVR**		
Test Period	**MAE**	**MAPE**	**RMSE**	**MAE**	**MAPE**	**RMSE**
OG	**243.34**	4.228E+16	**317.52**	245.25	**3.728E+16**	323.47
NY	237.07	5.443E+16	**309.41**	**233.36**	**4.925E+16**	310.04
JF	236.80	4.900E+16	307.82	**232.36**	**4.337E+16**	**306.60**
MT	238.13	4.833E+16	**307.76**	**234.68**	**4.171E+16**	308.48
SH	235.01	4.579E+16	**306.22**	**231.42**	**2.861E+16**	306.23

Table 4.13: Performance on Test Periods for Set **1+**

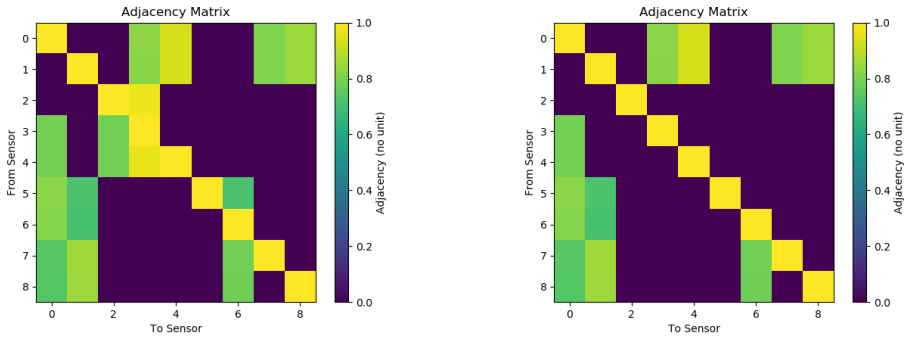
STGCN model achieves the best RMSE scores in 4 of my test periods.

4.6 Set 2 Experiments

4.6.1 Pruning

For Set **2**, I first compared the performance of models that used No Pruning and Same Junction Pruning. I chose $\epsilon = 0.6$ and $\delta = 1$ as my kernelization parameters, as these gave similar sparsity and value distribution to the best adjacency matrix found for Set **1**. These adjacency matrices are shown in Figure 4.18.

The results for models with each of these adjacency matrices on OG are shown in Table 4.14. Same Junction Pruning provided the best results for 2 of my metrics (MAE and MAPE) so I will use this in the comparison of my baseline model.



(a) Un-Pruned (b) Same Junction Pruning

Figure 4.18: Adjacency Matrix Pruning for Set **2**

Pruning Type	**MAE**	**MAPE**	**RMSE**
No Pruning	277.01	6.312E+16	**365.22**
Same Junction Pruning	**275.92**	**5.575E+16**	365.30

Table 4.14: Performance of Pruned Models for Set 2

4.6.2 Performance on OG Test Period

Table 4.15 shows the performance of my STGCN on Set **2** compared to the baseline SVR model. The STGCN outperforms my baseline in MAE and RMSE on every node and in the average. The SVR model shows better MAPE scores on 5 of the 9 nodes.

4.6.3 Performance on all Test Periods

Table 4.16 shows the results from all testing periods. My STGCN model performs very well, outperforming SVR in MAE on all periods, and in RMSE in 4 out for 5 periods.

	STGCN			**SVR**		
Node	**MAE**	**MAPE**	**RMSE**	**MAE**	**MAPE**	**RMSE**
0	**276.43**	5.929E+15	**371.10**	280.81	**5.914E+15**	375.55
1	**328.01**	**3.047E+16**	**423.53**	329.65	3.341E+16	425.83
2	**256.62**	3.064E+15	**338.92**	258.10	**2.777E+15**	341.95
3	**303.04**	**8.215E+15**	**397.60**	307.09	8.276E+15	404.17
4	**307.50**	**2.029E+16**	**397.05**	318.39	2.183E+16	411.73
5	**213.19**	3.136E+15	**289.00**	217.18	**2.946E+15**	296.05
6	**271.95**	2.936E+17	**369.51**	277.13	**2.715E+17**	379.34
7	**254.98**	**1.580E+16**	**353.67**	261.24	1.625E+16	362.56
8	**271.54**	1.213E+17	**347.34**	279.90	**1.027E+17**	362.32
Average	**275.92**	5.575E+16	**365.30**	281.06	**5.174E+16**	373.28

Table 4.15: Performance on OG for Set **2**

	STGCN			SVR		
Test Period	**MAE**	**MAPE**	**RMSE**	**MAE**	**MAPE**	**RMSE**
OG	**275.92**	5.575E+16	**365.30**	281.06	**5.174E+16**	373.28
NY	**262.10**	**6.342E+16**	**349.94**	265.14	6.49E+16	354.38
JF	**269.40**	5.796E+16	**357.89**	270.20	**5.648E+16**	359.35
MT	**262.59**	**5.888E+16**	**347.57**	265.21	6.007E+16	351.49
SH	**275.17**	5.848E+16	373.29	276.88	**5.296E+16**	**372.39**

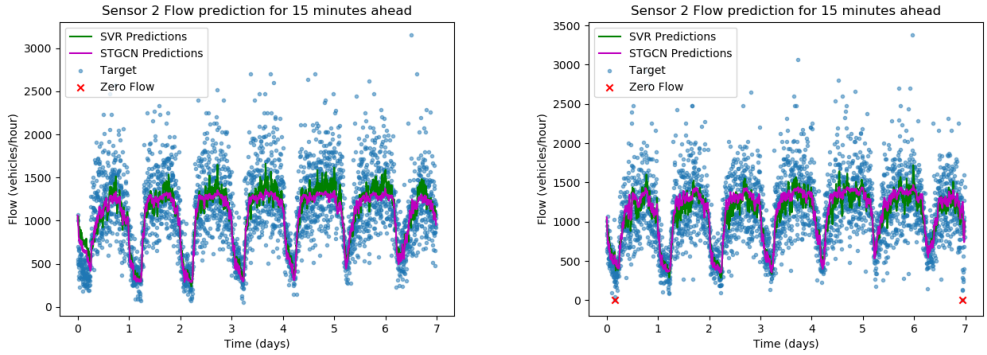
Table 4.16: Performance on Test Period for Set **2**

4.7 Sensor 196i performance in my test periods

When analysing my test period performance, I noticed that sensor 196i had an uncharacteristically low MAPE score compared to all other nodes and models I have observed. This is seen in both Set **1+** and Set **2**. Table 4.17 shows the performance on Node 2 of Set **2** (corresponding to sensor 196i) on my test periods. The MAPE score for NY and MT are orders of magnitude lower than the standard range I have found in other nodes and time periods (between 10^{15} and 10^{18}) We can look at the plots of my Node 2 predictions to understand this. In Figure 4.19 we see the true values and predicted values on Node 2 of Set **2** for two of my test periods. The NY period had one of the uncharacteristically low MAPE scores while JF had a MAPE score in the standard range. We can see in the plot that the NY period had no instances of zero flow, while the JF period had 2 (marked with red crosses).

	STGCN			SVR		
Test Period	**MAE**	**MAPE**	**RMSE**	**MAE**	**MAPE**	**RMSE**
Original	**256.62**	3.064E+15	**338.92**	258.10	**2.777E+15**	341.95
NY	**246.79**	**2.576E-01**	**328.24**	250.73	2.73E-01	331.92
JF	**248.23**	2.433E+15	**330.37**	248.47	**2.177E+15**	333.13
MT	**245.93**	2.478E-01	**324.10**	247.43	**2.435E-01**	330.71
SH	**257.71**	1.175E+15	**341.67**	257.83	**1.337E+15**	344.64

Table 4.17: Performance on Node 2 of Set **2** in my Test Periods



(a) Node 2 for Period NY

(b) Node 2 for Period JF

Figure 4.19: Predictions and true values on Node 2 in Set **2**

4.8 Comparing Common Sensors

Table 4.18 compares the performance of models from each set on their common sensors. The model trained on Set **1** performed particularly well, with the best MAE score in almost all of its shared junction (barring 48i and 145ii). In RMSE, my Set **1** model was beaten on 4 sensors by the Set **1+** model. In shared sensors between Set **1+** and Set **2**, the Set **2** model performed better in every case.

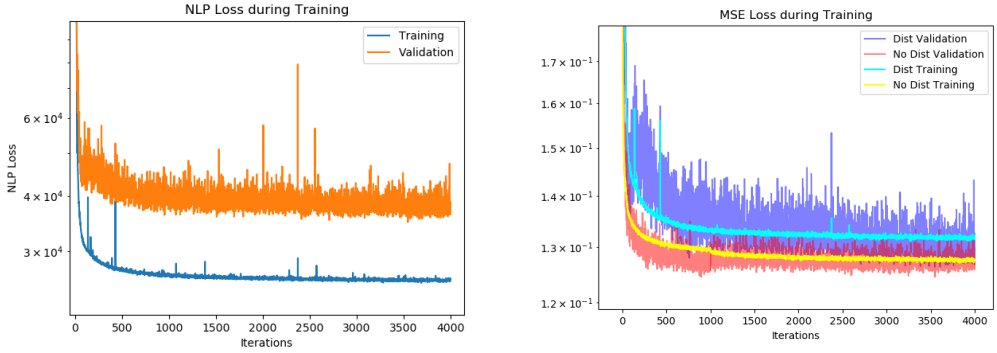
	Set **1**			Set **1+**			Set **2**		
Node	MAE	MAPE	RMSE	MAE	MAPE	RMSE	MAE	MAPE	RMSE
48i	**162.75**	**8.308E+15**	**213.40**	172.62	9.580E+15	221.07	-	-	-
48ii	218.84	1.606E+17	283.83	**216.39**	**1.599E+17**	**282.13**	-	-	-
48iii	**214.02**	**4.982E+16**	**273.03**	219.72	5.519E+16	280.02	-	-	-
145i	**169.03**	**8.975E+14**	250.18	169.86	8.977E+14	**249.35**	-	-	-
145ii	91.50	**1.477E+16**	120.54	**79.87**	2.010E+16	**107.17**	-	-	-
145iii	**288.50**	1.090E+17	**372.14**	292.55	**1.065E+17**	376.83	-	-	-
152i	**342.28**	1.346E+15	**436.10**	343.25	**1.305E+15**	436.17	-	-	-
152ii	**192.72**	**3.486E+15**	254.66	196.56	4.415E+15	**254.60**	-	-	-
152iii	**303.35**	4.411E+16	**388.79**	304.90	**4.309E+16**	391.45	-	-	-
183i	-	-	-	-	-	-	276.43	5.929E+15	371.10
183ii	-	-	-	-	-	-	328.01	3.047E+16	423.53
196i	-	-	-	258.02	3.337E+15	339.49	**256.62**	**3.064E+15**	**338.92**
196ii	-	-	-	312.67	8.423E+15	403.41	**303.04**	**8.215E+15**	**397.60**
196iii	-	-	-	311.41	2.295E+16	402.10	**307.50**	**2.029E+16**	**397.05**
288i	-	-	-	-	-	-	213.19	3.136E+15	289.00
288ii	-	-	-	-	-	-	271.95	2.936E+17	369.51
354i	**249.49**	**1.591E+16**	**346.50**	256.50	1.625E+16	353.39	254.98	**1.580E+16**	353.67
354ii	**267.83**	1.554E+17	**343.11**	272.45	1.399E+17	348.15	271.54	1.213E+17	347.34

Table 4.18: Performance of Different Set Models on Nodes

4.9 Estimate Distribution Experiment

4.9.1 Training

Figure 4.20 shows two images of training curves from training my distribution model. The first is in terms of NLP loss, the loss function it is training on. Although I am not training using MSE we can still measure it, as is shown in the second image, where it is compared against the training curves of my single output Set **1** model. These loss functions are based on normalized value and do not represent the MSE or NLP on de-normalized data.



(a) Node 2 for Period NY

(b) Node 2 for Period JF

Figure 4.20: Predictions and true values on Node 2 in Set **2**

4.9.2 Model Output

In this section I will look at the results of the estimate distribution design described in 3.6. I will refer to this as my NLP model and refer to previous single output models as MSE models. I will use Set **1** nodes for this section, using the pruning and kernelization parameters decided upon inSection 4.4 (Same Junction Pruning and $\epsilon = 1.5$). Figure 4.21 shows what the output looks like for a node in the OG period. The prediction window is between my outputted mean plus or minus my outputted standard deviation. This is $\mu_{i,n} \pm \sigma_{i,n}$ as described in Section 3.6 with $n = 0$ for Node 0 and i as the instances along the x-axis. Out NLP model contains 61,486 parameters, an increase of 1,089 from the MSE based model.

We can also plot the standard deviation values over time to understand how the uncertainty changes. Figure 4.22 shows the standard deviation plot for the same node

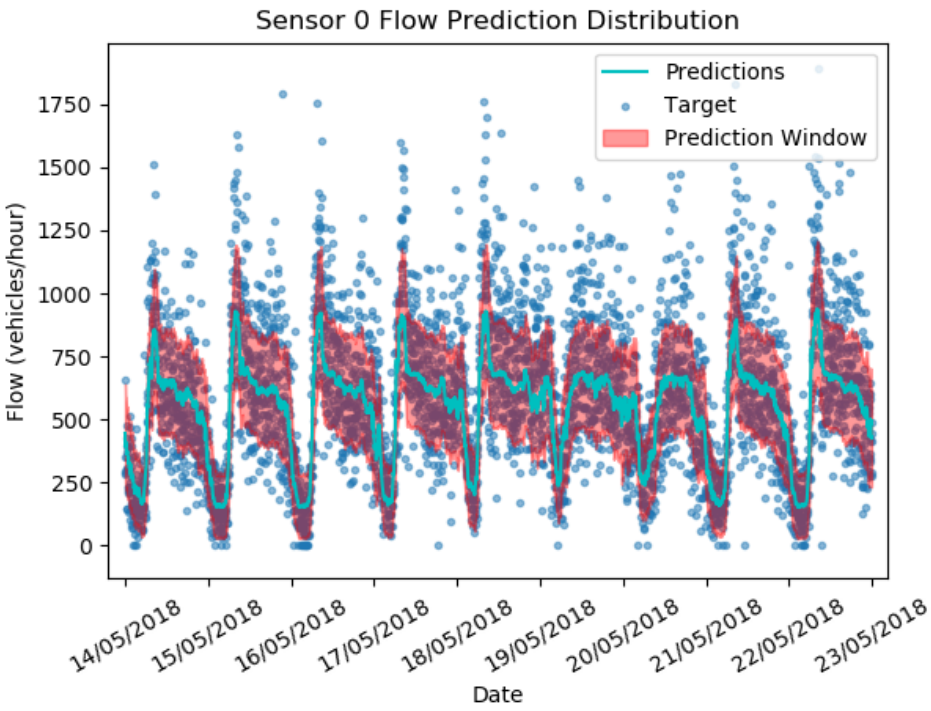


Figure 4.21: Estimate Distribution on OG for Node 0 of Set **1**

and time period as Figure 4.21. The estimate distribution and standard deviation plots for all nodes are shown in Appendix C.

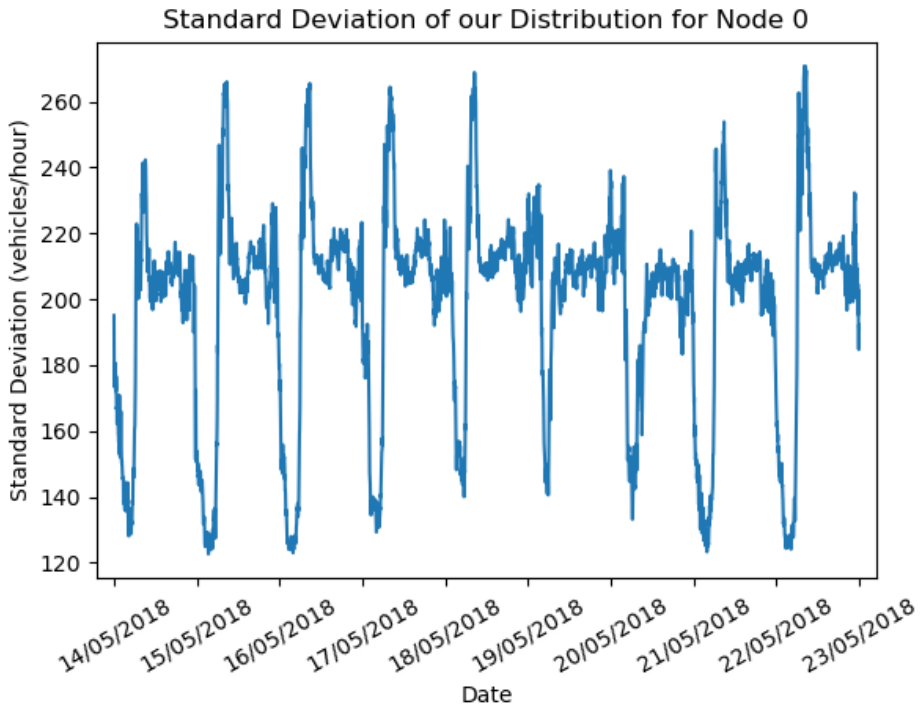


Figure 4.22: Standard Deviation on OG for Node 0 of Set **1**

In Figure 4.23 I show enlarged plots for one of the days in the OG period (15$^{\text{th}}$ May 2018). We can see that $\sigma_{i,0}$ follows a similar pattern to the $\mu_{i,0}$; starting to rise around 06:00, peaking around 08:00, falling to a more stable level from 09:00 to 21:00, and dipping lower between 23:00 and 06:00. However, there is some difference and the stable region and the late night dip. The standard distribution plot has a very flat stable region whereas, the flow prediction gradually decreases in this range. The dip into very low traffic occurs more abruptly in the standard deviation plot between 23:00 and 01:00 whereas the flow prediction drops more gradually between approximately 22:00 and 02:00.

4.9.3 Performance on OG Period

Performance Metrics

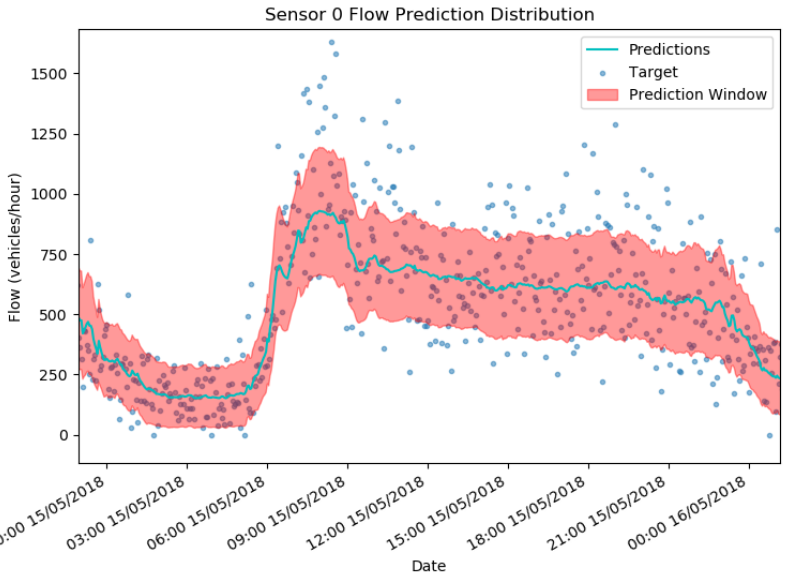
Table 4.19 shows how my NLP model performance compares to that of the MSE model on the OG period. The MSE model performs better on MAE and RMSE, though the differences are very small in a lot of cases.

	STGCN (MSE)			**STGCN (NLP)**		
Node	**MAE**	**MAPE**	**RMSE**	**MAE**	**MAPE**	**RMSE**
0	162.749	8.308E+15	**213.395**	**160.544**	**7.929E+15**	217.927
1	218.838	1.606E+17	283.827	**217.521**	**1.589E+17**	**282.511**
2	**214.016**	4.982E+16	**273.027**	230.854	**4.128E+16**	292.313
3	**169.027**	8.975E+14	**250.183**	169.442	**8.268E+14**	250.206
4	91.505	**1.477E+16**	120.541	**79.433**	2.107E+16	**107.573**
5	**288.503**	1.090E+17	**372.143**	304.187	**8.455E+16**	389.724
6	**342.279**	1.346E+15	**436.102**	342.490	**1.052E+15**	437.909
7	**192.716**	3.486E+15	**254.663**	193.453	**3.152E+15**	256.594
8	**303.347**	4.411E+16	**388.791**	305.467	**3.573E+16**	394.143
9	**249.494**	1.591E+16	**346.498**	262.613	**1.408E+16**	364.808
10	**267.829**	1.554E+17	**343.113**	283.382	**1.121E+17**	360.497
Average	**227.300**	5.124E+16	**298.389**	231.762	**4.370E+16**	304.928

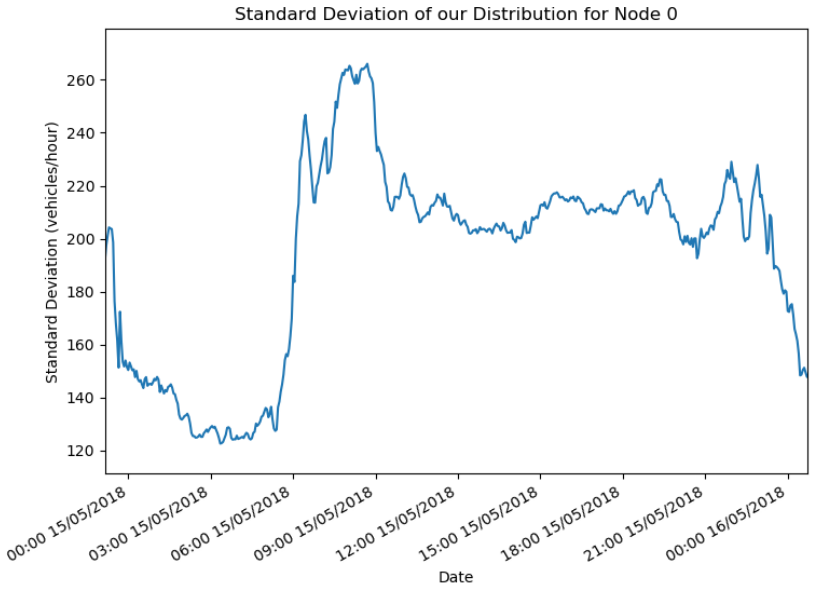
Table 4.19: Distribution Model Performance

Normal Distribution Characteristics

We can also measure the fraction of true values within the prediction window of $\mu_{i,n} \pm \sigma_{i,n}$. This fraction is shown for each node in the OG period in Table 4.20. We see that the average across my nodes is 0.6819 (with the greatest deviation from the



(a) Node 2 for Period NY



(b) Node 2 for Period JF

Figure 4.23: Flow and Standard Deviations on Node 0 for the 15$^{\text{th}}$ May 2018

average being 0.0528 for Node 3.

4.9.4 Performance on Test Periods

Table 4.21 shows the performance of my NLP model on the test sets. The MSE model performs better in very test period on MAE and RMSE. NLP achieves lower MAPE scores in every test period. The fraction of true values within the prediction

Node	Fraction within 1 σ
0	0.7072
1	0.6951
2	0.6106
3	0.7347
4	0.6988
5	0.6667
6	0.6868
7	0.6965
8	0.6838
9	0.6713
10	0.6488
Average	0.6819

Table 4.20: Fraction of True Values within the Window

	STGCN (MSE)			STGCN (NLP)			
Test	**MAE**	**MAPE**	**RMSE**	**MAE**	**MAPE**	**RMSE**	**Fraction within 1 σ**
OG	**227.300**	5.124E+16	**298.389**	231.762	**4.370E+16**	304.928	0.6819
NY	**224.048**	6.809E+16	**293.301**	230.11	**5.622E+16**	305.18	0.6705
JF	**220.800**	5.898E+16	**287.691**	226.23	**5.03E+16**	297.41	0.7002
MT	**221.545**	5.992E+16	**287.670**	226.67	**5.164E+16**	296.65	0.6927
SH	**215.583**	1.936E+05	**282.145**	220.20	**1.704E+05**	288.89	0.7093

Table 4.21: Performance of NLP model on Test Periods

window in each test set was similar to that found in the OG set.

4.10 One-Channel Experiments

4.10.1 Performance on OG Period

This section will look at forecasting the flow with only one channel of input (the flow) as opposed to the two channels used in my previous models (flow and density). For the same reasons as in Section 4.9, I will evaluate this method on my Set **1** junctions. This model used 59,821 parameters, a decrease of 567 from my original Set **1** model. Table 4.22 shows my one-channel model performance on the OG period, compared to that of my 2 channel model. The two-channel model performs best on average, and across most nodes, on all 3 metrics. However, one-channel achieves considerably better performance on Node 4 in terms of MAE and RMSE.

	STGCN (One Channels)			STGCN (Two Channels)		
Node	**MAE**	**MAPE**	**RMSE**	**MAE**	**MAPE**	**RMSE**
0	163.96	9.288E+15	217.82	**162.749**	**8.308E+15**	**213.395**
1	**217.16**	1.731E+17	**278.50**	218.838	**1.606E+17**	283.827
2	215.60	4.893E+16	274.23	**214.016**	4.982E+16	**273.027**
3	169.41	9.022E+14	**248.20**	**169.027**	**8.975E+14**	250.183
4	**81.48**	2.354E+16	**110.33**	91.505	**1.477E+16**	120.541
5	304.88	1.257E+17	389.73	**288.503**	**1.090E+17**	**372.143**
6	386.14	1.405E+15	487.62	**342.279**	**1.346E+15**	**436.102**
7	202.47	4.338E+15	261.34	**192.716**	**3.486E+15**	**254.663**
8	311.74	4.508E+16	400.52	**303.347**	**4.411E+16**	**388.791**
9	267.02	1.856E+16	361.24	**249.494**	**1.591E+16**	**346.498**
10	277.62	1.744E+17	351.74	**267.829**	**1.554E+17**	**343.113**
Average	236.13	5.684E+16	307.39	**227.300**	**5.124E+16**	**298.389**

Table 4.22: Performance of the One-Channel Model in OG on Set **1**

	STGCN (One Channels)			STGCN (Two Channels)		
Test	**MAE**	**MAPE**	**RMSE**	**MAE**	**MAPE**	**RMSE**
OG	236.13	5.684E+16	307.39	**227.300**	**5.124E+16**	**298.389**
NY	233.52	7.590E+16	304.95	**224.05**	**6.809E+16**	**293.30**
JF	230.65	6.875E+16	298.23	**220.80**	**5.898E+16**	**287.69**
MT	231.11	6.901E+16	298.60	**221.55**	**5.992E+16**	**287.67**
SH	225.32	2.332E+05	290.92	**215.58**	**1.936E+05**	**282.14**

Table 4.23: Performance of the One-Channel Model on the Test Periods

4.10.2 Performance on Test Periods

Table 4.23 shows how my one and two-channel models compare across the test periods. My two-channel model performs better in every metric across all test periods.

4.11 Discussion

4.11.1 Adjacency Matrix Kernelization

The best performing adjacency matrix used $\epsilon = 1.5$ and $\delta^2 = 10$ as its kernelization parameters. This gives a spread of adjacency values between $e^{-(1.5/10)} = 0.861$ and 1. This might not seem like a large range, but (41) uses values of $\epsilon = 0.6$ and $\delta^2 = 10$ giving a range between 0.942 and 1. $\epsilon = 1.5$ caused the kernelization to set 53 of the 121 adjacency values to zero. This gives us a graph with a sparsity of $53/121 = 43.8\%$ (or a density of 56.2%).

Unfortunately, I did not have time to perform comparisons of my δ value, or

comparisons of ϵ values in Set **2**

Although the diagonal matrix produced by $\epsilon = 0$ performed est in terms of MAPE, we saw in Section 4.7 that MAPE is not representative of performance. Since it performed worst in terms of MAE and RMSE, we can be sure that my spatial convolutions are having some effect.

4.11.2 Adjacency Matrix Pruning

Before my experiments I expected No Pruning to perform worst of all, and Unreasonable Rote Pruning to perform best. However, in my experiments Same Junction Pruning performed best, with No Pruning and Unreasonable Route Pruning achieving similar performance. I would say this is most likely due to incorrect choosing of my reasonable routes, rather than the technique itself. I chose them based on my understanding of the Dublin road network and possible routes a driver might take given a certain starting point. This could very easily be incorrect. Same Junction Pruning however, is formulaic and requires no human intervention. This means that it was able to perform well regardless of my judgement.

4.11.3 Set 1 Performance

Performance on OG by Node

Discounting the MAPE scores, my STGCN model performed better than my baseline on almost every node. The nodes on which it performed worse, were Nodes 1 4, and 6. These correspond to the second lowest, lowest, and highest flow rates in Set **1**. Node 6 had the highest autocorrelation values, and so I expected my SVR model to perform well on it. On the other hand, Node 1 was the node with the lowest autocorrelation, so I would have expected it to perform better with my STGCN model as it incorporates spatial information. This might indicate that the training of the SVR on individual nodes allowed for it to more accurately model the temporal patterns of outlier nodes. In my STGCN I are using the same temporal kernels for each node, meaning that they might optimize for nodes with flow rates in the most common range.

Performance on OG by Day

In the comparison of performance on days in the OG set, my STGCN model achieves a better MAE score for every day apart from Saturday and Sunday. One reason for this could be that my STGCN weights immediately previous intervals or intervals from previous days more highly than intervals from previous weeks, relative to my SVR. This might be remedied by changing the structure of my feature vector, including more data from previous weeks. I will discuss training on weekdays only in Section 4.11.8.

The RMSE scores for the daily breakdown appear to be out of skew with the MAE results. I suspect this is because of the way they are calculated in my SVR evaluation. Since SVR is run on each node individually, the daily RMSE is calculated for each node and then averaged after all nodes have been evaluated. This is shown in the equation below, with node n, instance i, number of nodes N, and number of instances in a day D.

$$\text{RMSE}_{\text{SVR}} = \frac{1}{N}\sum_{n=0}^{N-1}\left(\sqrt{\frac{1}{D}\sum_{i=0}^{D-1}(x_{i,n}-\hat{x}_{i,n})^2}\right) \tag{4.1}$$

The next equation shows how my STGCN days are evaluated. This is the correct method.

$$\text{RMSE}_{\text{STGCN}} = \frac{1}{ND}\left(\sqrt{\sum_{n=0}^{N-1}\sum_{i=0}^{D-1}(x_{i,n}-\hat{x}_{i,n})^2}\right) \tag{4.2}$$

I did not get the opportunity to go back and fix this, and so have avoided using this metric (though the daily breakdown for all models and test sets have been saved and are described in Appendix A). There is no issue for my MAE score since:

$$\frac{1}{N}\sum_{n=0}^{N-1}\left(\frac{1}{D}\sum_{i=0}^{D-1}|x_{i,n}-\hat{x}_{i,n}|\right) = \frac{1}{ND}\left(\sum_{n=0}^{N-1}\sum_{i=0}^{D-1}|x_{i,n}-\hat{x}_{i,n}|\right) \tag{4.3}$$

Performance on Test Periods

Our STGCN model performed better than my baseline on the OG and SH test sets. These were the closest test periods to my training data and so the traffic patterns

are likely to be more similar. The lesser performance on my NY, JF, and MT test periods, could be due to any combination of 3 things; overfitting, over-reliance on recent intervals, or changes in spatial patterns.

Overfitting would simply imply that I should use better regularization in my training in order to pick up on general patterns rather than those specific to a training period. I already suggested over-reliance on recent intervals as a cause for poor performance on my weekend tests. When moving to a new period it is important to take into account the weekly patterns, since these can be very different at different times of year. It could be that the case that the SVR model leverages these intervals to greater effect in its predictions. Lastly, it is likely that my spatial dependencies change throughout the year, meaning that my spatial convolution parameters are incorrectly trained for new periods. This could be fixed by training the model a larger range of time intervals. It is also possible that I could use different adjacency matrices depending on the time of year, though I am uncertain if the convolution parameters would be able to generalize to this.

When comparing the performance on my test sets, broken down by node, we saw a large variation in performance for Node 1. This node had the lowest values in the autocorrelation plot, and as such should be most difficult to predict by temporal dependencies. This could mean that the suitability of my spatial convolutions to the test period is what is largely determining the performance on Node 1. This supports the idea that the changes in spatial patterns are a large factor in how the model will perform on new data.

4.11.4 Set 1+ Performance

Performance on OG

The STGCN performs best in terms of MAE and RMSE on Set **1+** for period OG. When broken down by node we see that the SVR outperforms my STGCN in terms of MAE on Nodes 0, 2, 6, 7, and 12. Node 6 is the only one of these nodes that SVR did best in Set **1**. Node 6 was the node with the highest autocorrelation values, and so I would expect the SVR to perform well on it. I see no reason why SVR would perform better on the other nodes.

Performance on Test Periods

Our SVR performs best in 4 of my test periods, with regard to MAE, while STGCN achieves the highest RMSE score in 4 of the test periods. Each model is performing best on the loss function on which it was trained and so it is hard to compare them accurately.

4.11.5 Set 2 Performance

Pruning

For Set **2**, I only compared No Pruning and Same Junction Pruning. Same Junction Pruning performed better on MAE but worse on RMSE. Since the difference RMSE was much lower than that of MAE, it is fair to say that the Same Junction Pruning model is better.

Performance on OG

Our Set **2** STGCN performs better than my SVR on all nodes, both for MAE and RMSE. I did not expect the STGCN to outperform the SVR on this set, seeing as the junctions used appeared to be contemporaneous. I discuss the node performance again when comparing my sets in the next section.

Performance on Test Periods

Our STGCN achieves lower MAE and RMSE scores on almost all of my test sets (with the baseline performing better on RMSE for period SH. Again, I did not expect such good performance on this set. However, the quality of this model should be determined through comparisons of its shared nodes with the other sets.

4.11.6 Junction 196i & the MAPE scores

This shows that even a small number of zero values can skew the average MAPE score hugely, with the model that predicts lowest at these points achieving the best score. Upon reviewing other nodes and time periods I found that sensor 196i was the only sensor which had no instances of zero flow in any time period.

4.11.7 Relative Performance of the Sets

Despite my Set **2** model performing best, relative to its baseline, we see that in the nodes it shares with Set **1** are better predicted by the model trained on Set **1**. These are the nodes from Junction 354. The entries to this junction are at the edge of my map, and it is hard to see how they might be reached from my other nodes. However, there could be some spatial relationship involved with a node form Set **2**, that I do not understand.

Sets **1+** and **2** also share Junction 196, on which my Set **2** model performs best. The junctions in Set **2** that are not in Set **1+** are Junctions 183 and 288. Both of these have direct routes to Junction 196, meaning that my spatial convolution likely causes the difference in performance.

4.11.8 Training on Weekdays only

Since the weekends performed poorly on Set **1** it would be insightful to train and test my models on weekdays only. This might show an improvement in the relative power of my STGCN models over the SVR baseline. I created the functionality to generate weekday only input tensors and trained models for Sets **1**, **1+**, and **2** (details in Appendix A). Unfortunately, I was unable to create files to evaluate these models (or to train and evaluate SVR models on weekdays only) before the submission of this dissertation.

4.11.9 Estimate Distribution Performance

Training

From my training curves we can see that the NLP model does not take longer to train than for my MSE model, though it does not reach as low a minimum as my MSE model. My NLP training curves contain more sharp spikes than we would see on an MSE loss curve, which tends to have positive and negative noise.

Performance of the Predictions

The predictions μ of my NLP model perform worse than my MSE model on almost every node in OG, for both MAE and RMSE. Nodes 1, and 4, on which it performs

better on, are the two smallest nodes in my set, and as such are not well representative of model quality. The MSE model also performs better on all other test periods.

Performance of the Prediction Window

We stated in Section 3.6 that having approximately 68.27% of the true values within one standard deviation of the mean would show that the distribution was well calibrated for the uncertainty of my model. Not only was this the case for all my nodes in OG, but it is also the case for all of the test periods (with an average of 69.09%). It is quite impressive that the model is able to generalize this capability to unseen periods.

Our standard deviation plots are also very interesting. These plots (as seen in the plots in Section 4.9 and Appendix C) are generally more discrete than the flow predictions, with flatter areas and more abrupt changes. This could help us to distinguish the different modes of traffic happening throughout the day.

4.11.10 One-Channel Performance

Our one-channel model performed worse than the 2 channel model on all nodes in the OG period (except Nodes 1 and 4 which I discussed earlier). The difference in some places is much larger than in others. The greatest difference is on Node 6 (the Node with the highest average flow) where the difference in MAE was 43.87. This, however, is an outlier, with the performance on most nodes being much lower. The two-channel model performed better on all of the test periods, with a difference of around 10 in both MAE and RMSE.

Chapter 5

Conclusion

5.1 Research Findings and Critical Assessments

In this dissertation, I applied a GNN model to traffic forecasting in an urban setting, something that is rarely done in the field. Moreover, this is the first use of a GNN to forecast traffic in Dublin. The standard data sets used in traffic forecasting are usually highway based, such as the BERJ4 and PeMSD7 datasets used in the paper on the original design of the STGCN model (41). City traffic is harder to forecast due to relatively greater noise, so this dissertation provides a reference for how an STGCN model can perform in an urban setting.

The thorough description of my design's inner workings will allow other researchers to easily understand and carry on this work. Since the designs described in academic papers can be at a very high level, it is very valuable to have a description at a finer scale. I detail the inner workings of each component along with the form the data takes at every point. Hopefully, this will allow someone who is new to GNNs to quickly understand the functionality of the STGCN design.

I implemented a baseline SVR model, similar to that used in (5), for comparison with my STGCN model. I found that the STGCN outperformed the baseline in most cases. My SVR model operated on nodes individually meaning that it was only able to leverage temporal dependencies in its predictions. This comparison allowed me to demonstrate that the spatial graph convolutions used were indeed having a positive effect in performance.

I suspect the feature vector used in my model may not be optimal for my STGCN model. Two factors might have achieved better results; data from an increased number of days and weeks previous, and longer sequences of intervals from these previous days and weeks. Longer sequences of intervals from a given period would likely play to the strength of the temporal convolutions and show improved performance relative to the baseline. The baseline uses elements of the feature vector independently, disregarding the sequence in which they come, meaning that it would be less affected by this.

My primary model incorporates flow and density channels in my input data. No GNN model I found used both of these channels for its predictions, though some used secondary channels representing other factors. It was unclear whether this would aid with prediction, so to investigate this I trained a model on flow alone. This comparison showed that the density did indeed aid in prediction, with my two-channel model achieving consistently better results.

I also created a model which outputs a normally distributed estimate window around the prediction. This is not something I had seen done in the Deep Learning traffic forecasting literature. Although this led to worse predictions of the true values, the plots of the standard deviation over time could offer valuable insights into the data. The consistency of the normal distribution in capturing around 68% of the true values within 1σ was very promising and generalized well to unseen data.

Unfortunately, I did not perform sufficient cross validation on the adjacency matrices of each node set to establish the best performance. Seeing as the adjacency matrix is so integral to the strength of the STGCN, it is vital that we use the one that most accurately depicts the traffic network.

5.2 Limitations

I only had access to SCATS data between January 1^{st} 2018 and June 30^{th} 2018. This limited the training and testing periods used for my model. Training on a whole year of data would have allowed my model to generalize better to the test sets throughout the year.

I was also limited by the computing power of the college computers and my

personal computer. Training for 1,000 epochs took approximately 3 hours on the college computers and 7 hours on my personal computer. Using a device with a better GPU would have allowed me to train more models for comparison.

5.3 Recommendations for Future Work

I recommend improving upon the adjacency matrix design, through both kernelization parameters and pruning. For each set I would suggest performing a more granular cross validation of the kernelization parameters. A more detailed analysis of the Dublin road network would lead to a more accurate depiction of the connection between nodes than the Unreasonable Route Pruning I tested in my model.

I also suggest scaling up the size of the data set, both in terms of the number of nodes, and the time periods used. Increasing the number of nodes is an obvious step to better compare how the model compares to current methods used in Dublin, but I believe it would also improve the relative performance of the STGCN. By having many more interconnected nodes we would expect the model to be able to better leverage the spatial data. Increasing the time span used for training and testing would also give us a more accurate indication of the STGCN's performance, and allow it to generalize better to unseen time periods and aberrant behaviour, such as bank holidays.

Lastly, I believe that changes to the feature vector would have profound effects, and so recommend a cross validation of models that use. We saw in the methodology that increasing the feature vector size would not hugely alter the number of parameters, and as such not cause too great an increase in training time. With data from a longer time span, it would be possible to use more data from previous weeks in the feature vector without cutting down the data we can use for training and testing by too much. I would also suggest trialling increasing the length of the continuous periods in the features (from immediately previous, and from previous days and weeks), as I believe that this would allow for the temporal convolutions to perform better. Altering the feature vector is easily done and simply requires that we change the relative index list.

Bibliography

[1] Al-Saffar, A. A. M., Tao, H., and Talab, M. A. Review of deep convolution neural network in image classification. In *2017 International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET)* (2017), pp. 26–31.

[2] Bahdanau, D., Cho, K. H., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *3rd International Conference on Learning Representations, ICLR 2015* (1 2015). 3rd International Conference on Learning Representations, ICLR 2015 ; Conference date: 07-05-2015 Through 09-05-2015.

[3] Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and deep locally connected networks on graphs. *2nd International Conference on Learning Representations, ICLR 2014* (2014).

[4] Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., and Varoquaux, G. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning* (2013), pp. 108–122.

[5] Chen, R., Liang, C.-Y., Hong, W.-C., and Gu, D. Forecasting holiday daily tourist flow based on seasonal support vector regression with adaptive genetic algorithm. *Applied Soft Computing 26* (12 2014).

[6] Cong, Y., Wang, J., and Li, X. Traffic flow forecasting by a least squares support vector machine with a fruit fly optimization algorithm. *Procedia Engi-*

neering 137 (2016), 59–68.

[7] Council, D. C. *Traffic Volumes from SCATS Traffic Management System - Datasets*, 2021. https://data.smartdublin.ie/dataset/dcc-scats-detector-volume-jan-jun-2020 [Accessed: 2021-12-09].

[8] Dakic, I., and Stevanovic, A. On development of arterial fundamental diagrams based on surrogate density measures from adaptive traffic control systems utilizing stop-line detection. *Transportation Research Procedia 23* (2017), 942–961. Papers Selected for the 22nd International Symposium on Transportation and Traffic Theory Chicago, Illinois, USA, 24-26 July, 2017.

[9] Dakic, I., and Stevanovic, A. On development of arterial fundamental diagrams based on surrogate density measures from adaptive traffic control systems utilizing stop-line detection. *Transportation Research Part C: Emerging Technologies 94* (2018), 133–150.

[10] Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems 29* (2016), 3844–3852.

[11] Derrow-Pinion, A., She, J., Wong, D., Lange, O., Hester, T., Perez, L., Nunkesser, M., Lee, S., Guo, X., Wiltshire, B., et al. Eta prediction with graph neural networks in google maps. *Proceedings of the 30th ACM International Conference on Information & Knowledge Management* (2021), 3767–3776.

[12] Fang, S., Zhang, Q., Meng, G., Xiang, S., and Pan, C. Gstnet: Global spatial-temporal network for traffic flow prediction. *IJCAI* (2019), 2286–2293.

[13] Feliciani, C., Murakami, H., and Nishinari, K. A universal function for capacity of bidirectional pedestrian streams: Filling the gaps in the literature. *PLOS ONE 13* (12 2018), e0208496.

[14] Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. Convolutional sequence to sequence learning. *Proceedings of the 34th International Conference on Machine Learning 70* (12 2017), 1243–1252.

[15] Geng, X., Li, Y., Wang, L., Zhang, L., Yang, Q., Ye, J., and Liu, Y. Spatiotemporal multi-graph convolution network for ride-hailing demand forecasting.

[16] Gori, M., Monfardini, G., and Scarselli, F. A new model for earning in graph domains. *Proceedings of the International Joint Conference on Neural Networks 2* (2005), 729–734.

[17] Guo, K., Hu, Y., Qian, Z., Liu, H., Zhang, K., Sun, Y., Gao, J., and Yin, B. Optimized graph convolution recurrent neural network for traffic prediction. *IEEE Transactions on Intelligent Transportation Systems 22* (2021), 1138–1149.

[18] Guo, S., Lin, Y., Feng, N., Song, C., and Wan, H. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence 33* (07 2019), 922–929.

[19] Hammond, D. K., Vandergheynst, P., and Gribonval, R. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis 30*, 2 (2011), 129–150.

[20] Inc., U. T. *H3: Uber's Hexagonal Hierarchical Spatial Index*, 2022. https://eng.uber.com/h3/ [Accessed: 2022-04-09].

[21] INRIX. *Dublin's INRIX Scorecard Report*, 2021. https://inrix.com/scorecard-city/?city=Dublin&index=16 [Accessed: 2021-12-09].

[22] INRIX. *Global Traffic Scorecard 2021 | INRIX Global Traffic Rankings*, 2021. https://inrix.com/scorecard/ [Accessed: 2021-12-09].

[23] Johansson, U., Boström, H., Löfström, T., and Linusson, H. Regression conformal prediction with random forests. *Machine Learning 97* (2014), 155–176.

[24] Jumper, J. M., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard,

A., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., Back, T., Petersen, S., Reiman, D. A., Clancy, E., Zielinski, M., Steinegger, M., Pacholska, M., Berghammer, T., Bodenstein, S., Silver, D., Vinyals, O., Senior, A. W., Kavukcuoglu, K., Kohli, P., and Hassabis, D. Highly accurate protein structure prediction with alphafold. *Nature 596* (2021), 583 – 589.

[25] Lana, I., Ser, J. D., Velez, M., and Vlahogianni, E. I. Road traffic forecasting: Recent advances and new challenges. *IEEE Intelligent Transportation Systems Magazine 10* (6 2018), 93–109.

[26] Li, Y., Yu, R., Shahabi, C., and Liu, Y. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *International Conference on Learning Representations* (2018).

[27] Lu, Z., Lv, W., Xie, Z., Du, B., and Huang, R. Leveraging graph neural network with lstm for traffic speed prediction. *2019 IEEE SmartWorld* (2019), 74–81.

[28] Mai, W., Chen, J., and Chen, X. Time-evolving graph convolutional recurrent network for traffic prediction. *Applied Sciences 12*, 6 (2022).

[29] Micheli, A. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks 20* (2009), 498–511.

[30] Moorthy, C. K., and Ratcliffe, B. G. Short term traffic forecasting using time series methods. *Transportation Planning and Technology 12* (1988), 45–56.

[31] Okutani, I., and Stephanedes, Y. J. Dynamic prediction of traffic volume through kalman filtering theory. *Transportation Research Part B: Methodological 18* (1984), 1–11.

[32] Quiñonero-Candela, J., Rasmussen, C., Sinz, F., Bousquet, O., and Schölkopf, B. *Evaluating Predictive Uncertainty Challenge*, vol. 3944. 04 2006, pp. 1–27.

[33] Song, C., Lin, Y., Guo, S., and Wan, H. Spatial-temporal synchronous graph convolutional networks: A new framework for spatial-temporal network data forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence 34* (2020), 914–921.

[34] Tang, J., Wang, Y., Wang, H., Zhang, S., and Liu, F. Dynamic analysis of traffic time series at different temporal scales: A complex networks approach. *Physica A: Statistical Mechanics and its Applications 405* (7 2014), 303–315.

[35] Tang, J., Zou, Y., Ash, J., Zhang, S., Liu, F., and Wang, Y. Travel time estimation using freeway point detector data based on evolving fuzzy neural inference system. *PLoS ONE 11* (2 2016).

[36] ting Zhong, J., and Ling, S. *Key Factors of K-Nearest Neighbours Non-parametric Regression in Short-Time Traffic Flow Forecasting.* IEEE, 12 2015, pp. 9–12.

[37] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. Wavenet: A generative model for raw audio, 2016.

[38] Wang, S., Li, Y., Zhang, J., Meng, Q., Meng, L., and Gao, F. Pm2.5-gnn: A domain knowledge enhanced graph neural network for pm2.5 forecasting. pp. 163–166.

[39] Wang, Y. Graph neural network in traffic forecasting: A review. *2021 the 3rd International Conference on Robotics Systems and Automation Engineering (RSAE)* (2021), 34–39.

[40] Yin, X., Wu, G., Wei, J., Shen, Y., Qi, H., and Yin, B. Deep learning on traffic prediction: Methods, analysis and future directions. *IEEE Transactions on Intelligent Transportation Systems* (2021).

[41] Yu, B., Yin, H., and Zhu, Z. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. vol. 2018-July, International Joint Conferences on Artificial Intelligence, pp. 3634–3640.

[42] Zhang, J., Zheng, Y., and Qi, D. Deep spatio-temporal residual networks for citywide crowd flows prediction.

[43] Zhang, J., Zheng, Y., Qi, D., Li, R., and Yi, X. Dnn-based prediction model for spatio-temporal data. *GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems* (10 2016).

[44] Zheng, C., Fan, X., Wang, C., and Qi, J. Gman: A graph multi-attention network for traffic prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2020), vol. 34, pp. 1234–1241.

[45] Zheng, Z., Yang, Y., Liu, J., Dai, H. N., and Zhang, Y. Deep and embedded learning approach for traffic flow prediction in urban informatics. *IEEE Transactions on Intelligent Transportation Systems 20* (10 2019), 3927–3939.

Appendices

Appendix A

Description of my Codebase

A.1 Codebase

The code used for the STGCN design and analysis code can be found on GitHub: https://github.com/oscarcrowley1/thesis.

In my code BRAVO and BRAVOPLUS refer to Sets **1** and **1+**, while ALPHA refers to Set **2**

The `interpret_csv_xxxx` folders contain the files for creating our node value tensors and adjacency matrices for each set. These are zipped into a `SCATS_xxxx` which is copied into our `DC_STGCN` folder for use in training.

The `DC_STGCN` folder contains the files for creating and training our STGCN models.

The `analysis` folder contains the files for evaluating our model performance.

The `saved_models` folder contains all models I have run, with the `runs` folder containing the TensorBoard files relating to the training of each one.

The `svr` folder contains the code for training and running our SVR model.

More detailed description of the codebase and how to run the files is found in the `readme.txt` file. This file also contains a list of tools and libraries used.

A.2 Models

The models used in Section 4 are saved in the folder `final_models`. They are each saved within folders along with their training and evaluation data.

- alpha_d03_nodist_2403_PRUNING: Set **2**, Same Junction Pruning, 1000 epochs
- alpha_d03_nodist_2403: Set **2**, Same Junction Pruning, 1000 epochs
- alpha_d03_nodist_2403_P_weekdays: Set **2**, Same Junction Pruning, trained on weekdays only, 1000 epochs
- bravo_d03_nodist_2903_diag: Set **1**, Epsilon 0, No Pruning, 1000 epochs
- bravo_d03_nodist_2403_e06d10: Set **1**, Epsilon 0.6, No Pruning, 1000 epochs
- bravo_d03_nodist_2503_e15d10: Set **1**, Epsilon 1.5, No Pruning, 1000 epochs
- bravo_d03_nodist_2503_e26d10: Set **1**, Epsilon 2.6, No Pruning, 1000 epochs
- bravo_d03_i4000_nodist_2503_e15d10_PRUNING: Set **1**, Epsilon 1.5, Same Junction Pruning, 4000 epochs
- bravo_d03_nodist_2503_e15d10_PRUNING: Set **1**, Epsilon 1.5, Same Junction Pruning, 1000 epochs
- bravo_d03_nodist_2503_e15d10_MOREPRUNING: Set **1**, Epsilon 1.5, Unreasonable Route Pruning, 1000 epochs
- bravo_d03_nodist_e1510_P_weekdays: Set **1**, Epsilon 1.5, Same Junction Pruning, 1000 epochs, trained on weekdays only
- bravoplus_d03_nodist_2503_e15d10_PRUNING: Set **1+**, Epsilon 1.5, Same Junction Pruning, 1000 epochs
- bravoplus_d03_nodist_2503_e15d10_P_weekdays: Set **1+**, Epsilon 1.5, Same Junction Pruning, 1000 epochs, trained on weekdays only
- bravo_d03_DIST_2503_e15d10_PRUNING: Set **1**, Estimate Distribution Model, Epsilon 1.5, Same Junction Pruning, 1000 epochs
- bravo_d03_DIST_2503_e15d10_PRUNING_longrun: Set **1**, Estimate Distribution Model, Epsilon 1.5, Same Junction Pruning, 5000 epochs

- bravo_d03_FLOW_nodist_2503_e15d10_PRUNING: Set **1**, One Channel Flow Only Model, Epsilon 1.5, Same Junction Pruning, 1000 epochs

Appendix B

Code

B.1 Codebase

The code used for the STGCN design and analysis code can be found on GitHub: https://github.com/oscarcrowley1/thesis.

In my code BRAVO and BRAVOPLUS refer to Sets **1** and **1+**, while ALPHA refers to Set **2**

The `interpret_csv_xxxx` folders contain the files for creating our node value tensors and adjacency matrices for each set. These are zipped into a `SCATS_xxxx` which is copied into our `DC_STGCN` folder for use in training.

The `DC_STGCN` folder contains the files for creating and training our STGCN models.

The `analysis` folder contains the files for evaluating our model performance.

The `saved_models` folder contains all models I have run, with the `runs` folder containing the TensorBoard files relating to the training of each one.

The `svr` folder contains the code for training and running our SVR model.

More detailed description of the codebase and how to run the files is found in the `readme.txt` file. This file also contains a list of tools and libraries used.

B.2 Models

The models used in Section 4 are saved in the folder `final_models`. They are each saved within folders along with their training and evaluation data.

- alpha_d03_nodist_2403_PRUNING: Set **2**, Same Junction Pruning, 1000 epochs
- alpha_d03_nodist_2403: Set **2**, Same Junction Pruning, 1000 epochs
- alpha_d03_nodist_2403_P_weekdays: Set **2**, Same Junction Pruning, trained on weekdays only, 1000 epochs
- bravo_d03_nodist_2903_diag: Set **1**, Epsilon 0, No Pruning, 1000 epochs
- bravo_d03_nodist_2403_e06d10: Set **1**, Epsilon 0.6, No Pruning, 1000 epochs
- bravo_d03_nodist_2503_e15d10: Set **1**, Epsilon 1.5, No Pruning, 1000 epochs
- bravo_d03_nodist_2503_e26d10: Set **1**, Epsilon 2.6, No Pruning, 1000 epochs
- bravo_d03_i4000_nodist_2503_e15d10_PRUNING: Set **1**, Epsilon 1.5, Same Junction Pruning, 4000 epochs
- bravo_d03_nodist_2503_e15d10_PRUNING: Set **1**, Epsilon 1.5, Same Junction Pruning, 1000 epochs
- bravo_d03_nodist_2503_e15d10_MOREPRUNING: Set **1**, Epsilon 1.5, Unreasonable Route Pruning, 1000 epochs
- bravo_d03_nodist_e1510_P_weekdays: Set **1**, Epsilon 1.5, Same Junction Pruning, 1000 epochs, trained on weekdays only
- bravoplus_d03_nodist_2503_e15d10_PRUNING: Set **1+**, Epsilon 1.5, Same Junction Pruning, 1000 epochs
- bravoplus_d03_nodist_2503_e15d10_P_weekdays: Set **1+**, Epsilon 1.5, Same Junction Pruning, 1000 epochs, trained on weekdays only
- bravo_d03_DIST_2503_e15d10_PRUNING: Set **1**, Estimate Distribution Model, Epsilon 1.5, Same Junction Pruning, 1000 epochs
- bravo_d03_DIST_2503_e15d10_PRUNING_longrun: Set **1**, Estimate Distribution Model, Epsilon 1.5, Same Junction Pruning, 5000 epochs

- bravo_d03_FLOW_nodist_2503_e15d10_PRUNING: Set **1**, One Channel Flow Only Model, Epsilon 1.5, Same Junction Pruning, 1000 epochs

Appendix C

Distribution Model Plots

C.1 Estimate Distribution Plots

C.1.1 Estimates and Estimate Windows

These plots show the estimate and estimate window (within 1 σ) of each node for our OG test period

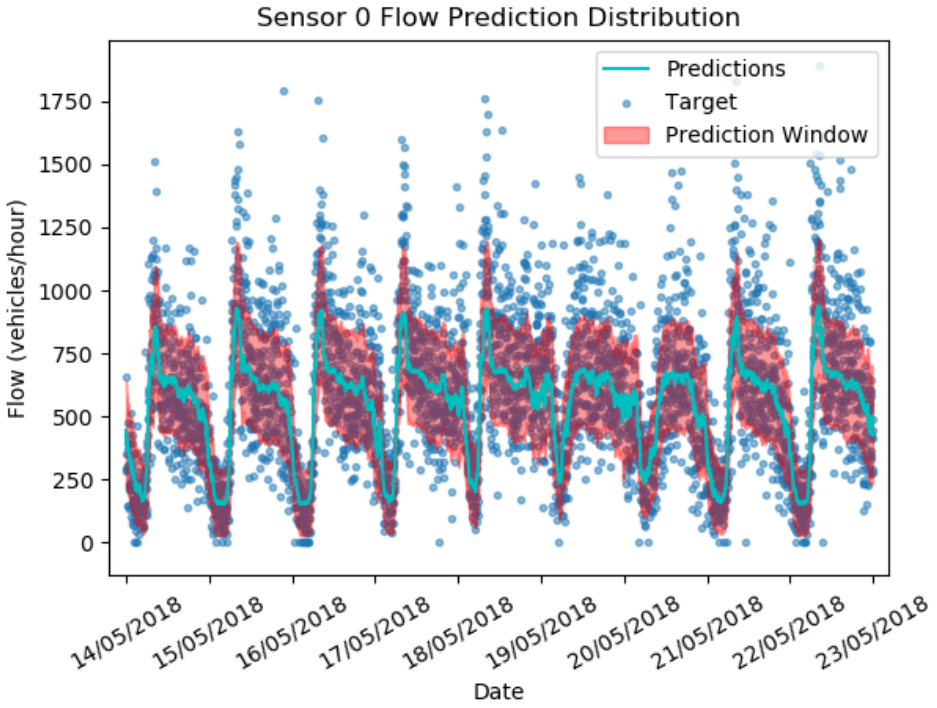


Figure C.1: Node 0

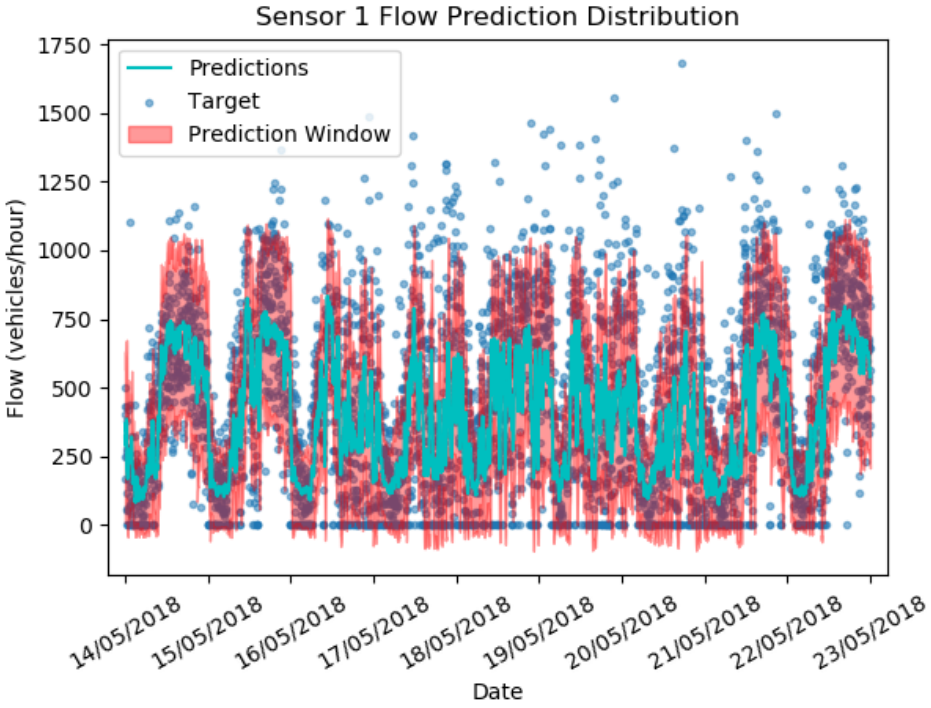


Figure C.2: Node 1

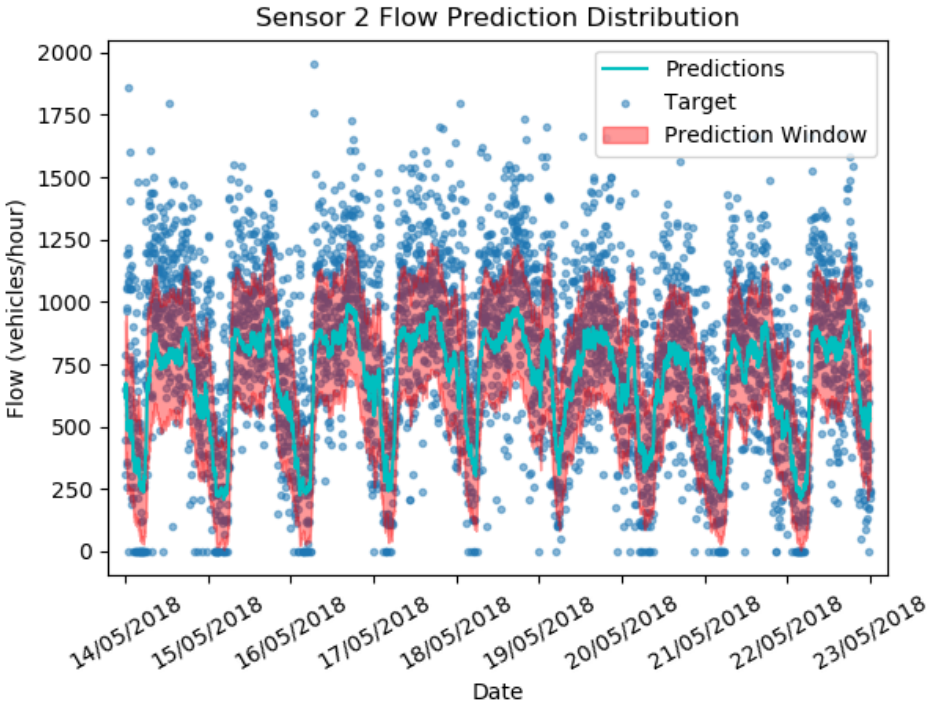


Figure C.3: Node 2

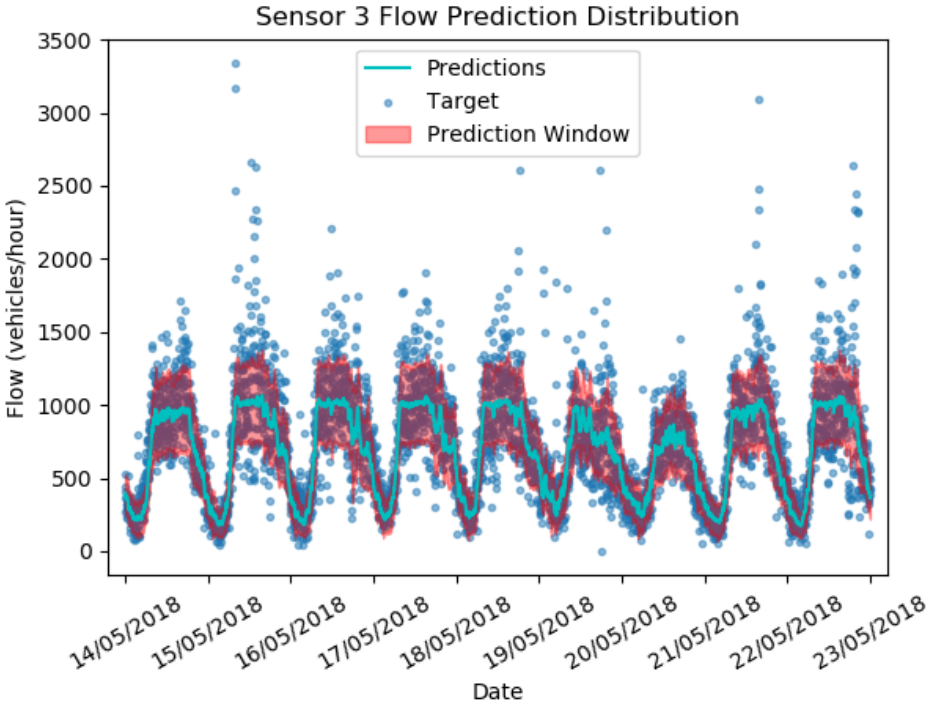


Figure C.4: Node 3

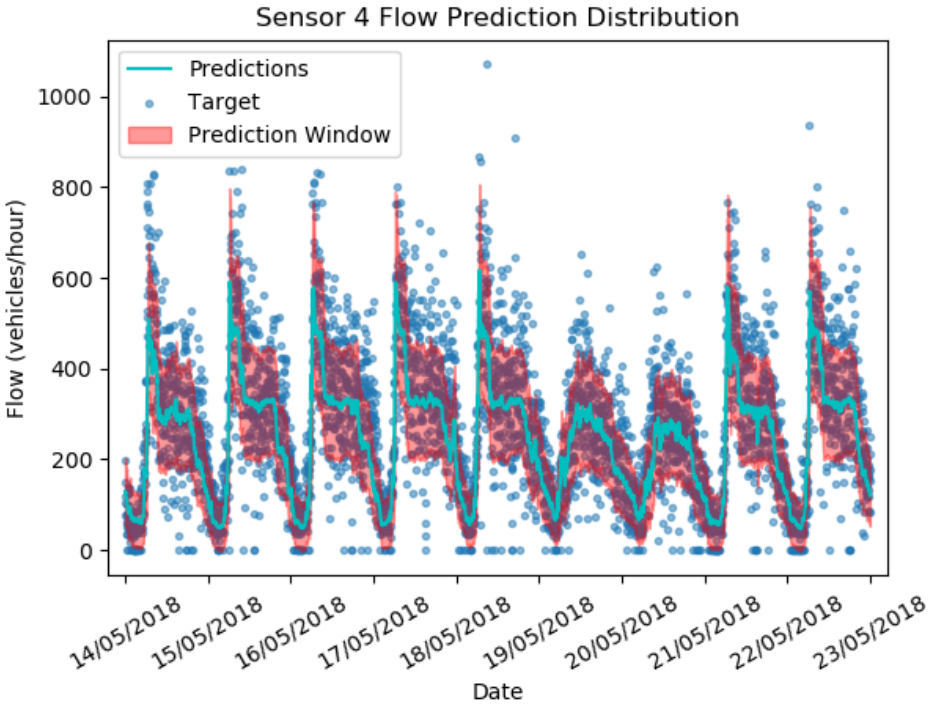


Figure C.5: Node 4

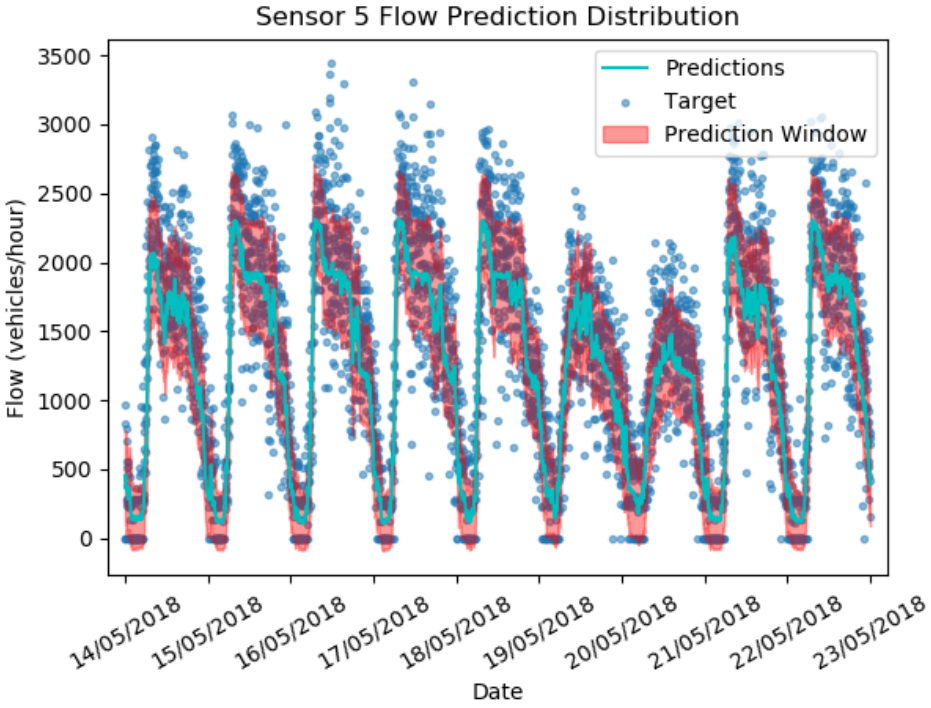


Figure C.6: Node 5

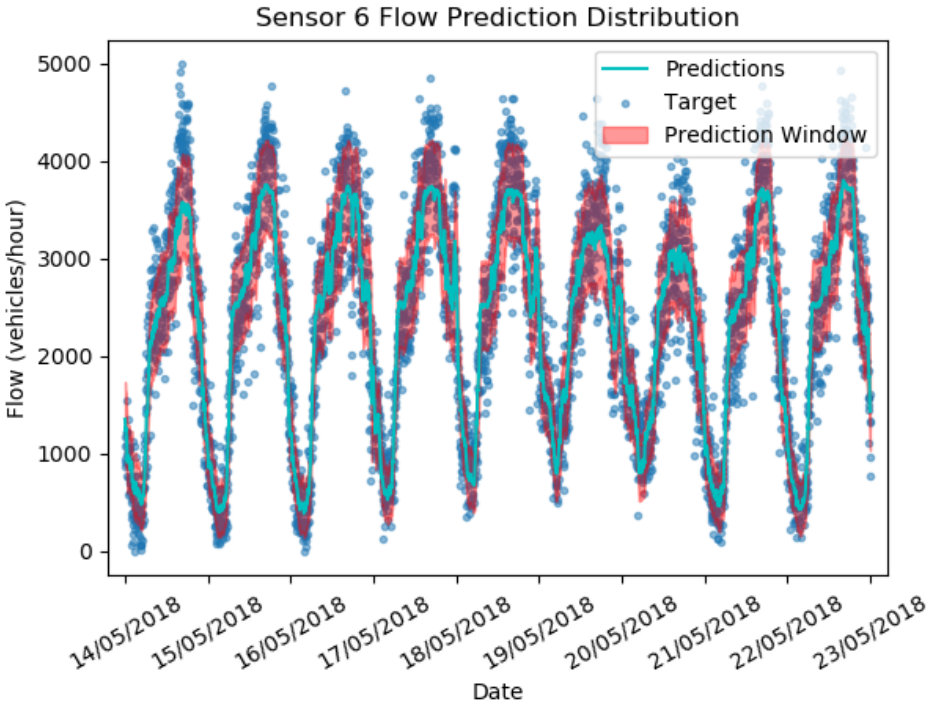


Figure C.7: Node 6

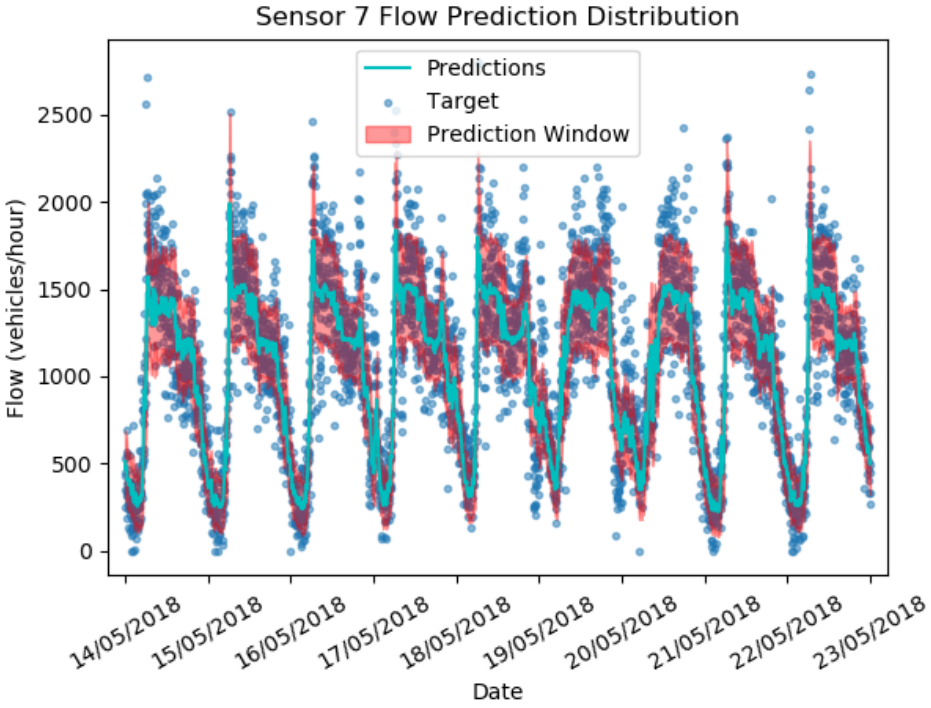


Figure C.8: Node 7

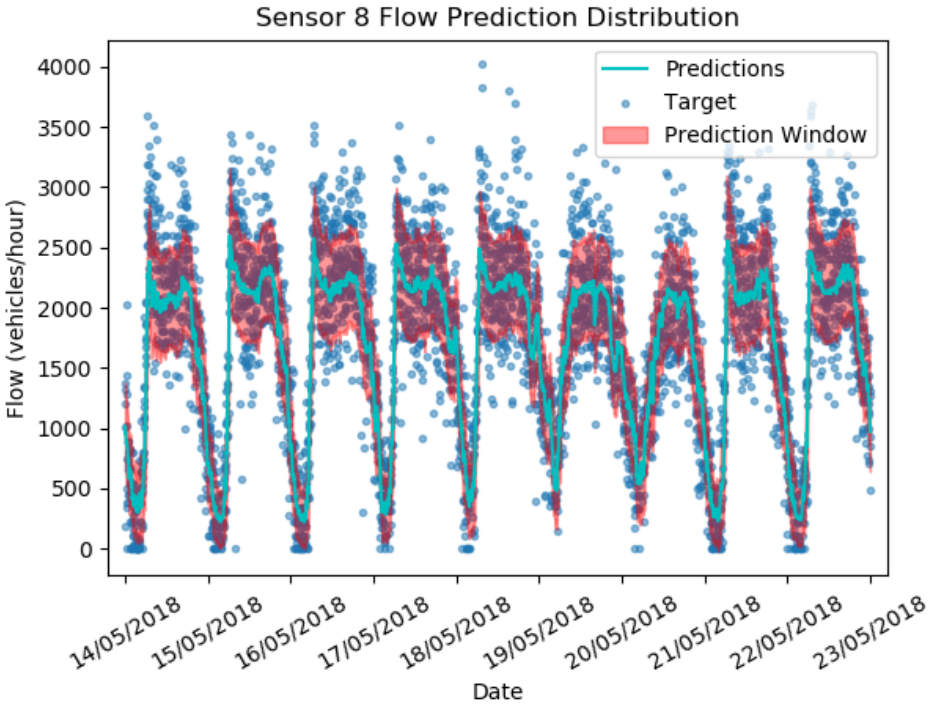


Figure C.9: Node 8

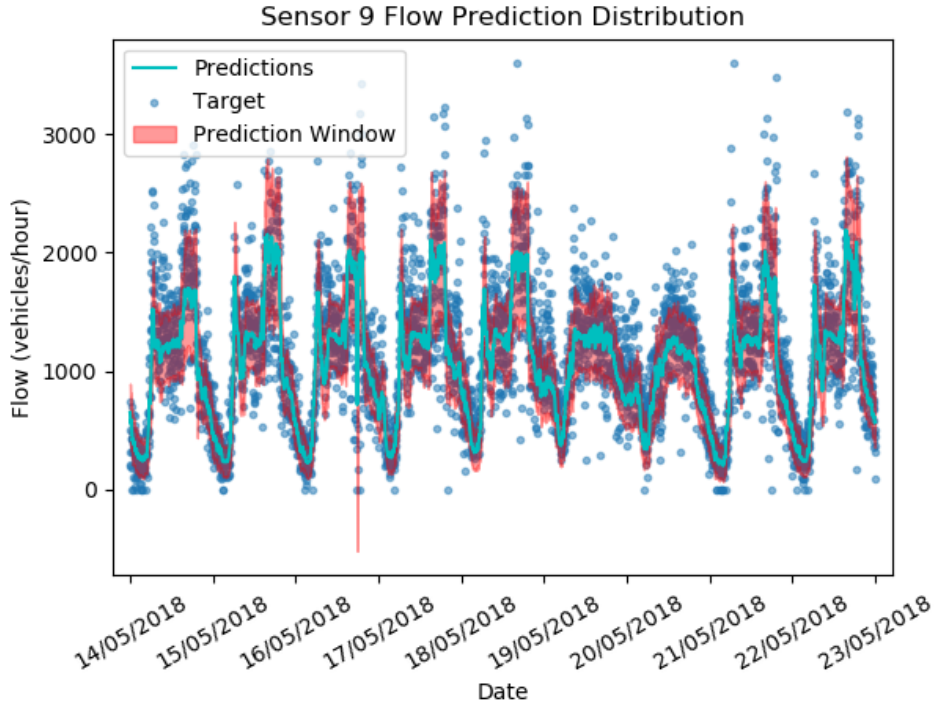


Figure C.10: Node 9

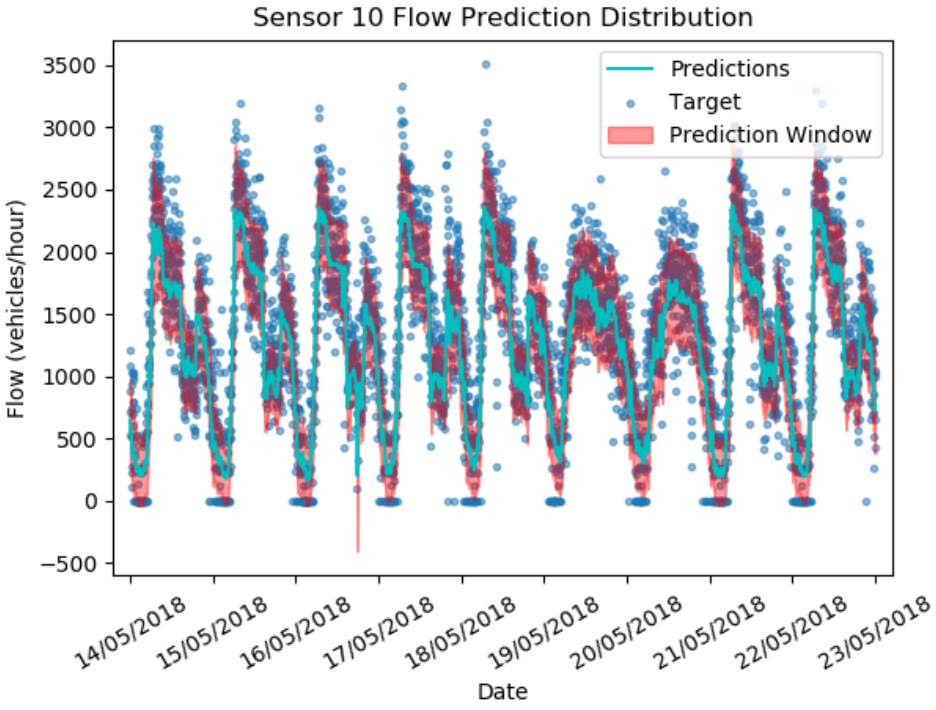


Figure C.11: Node 10

C.2 Standard Deviations

These plots show the standard at each node for our OG test period

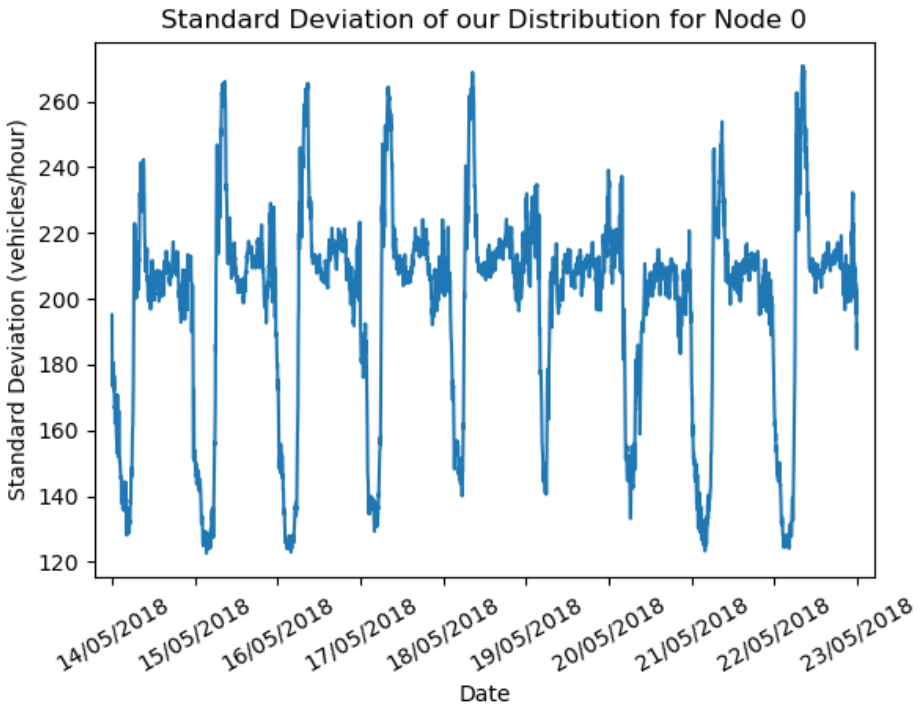


Figure C.12: Node 0

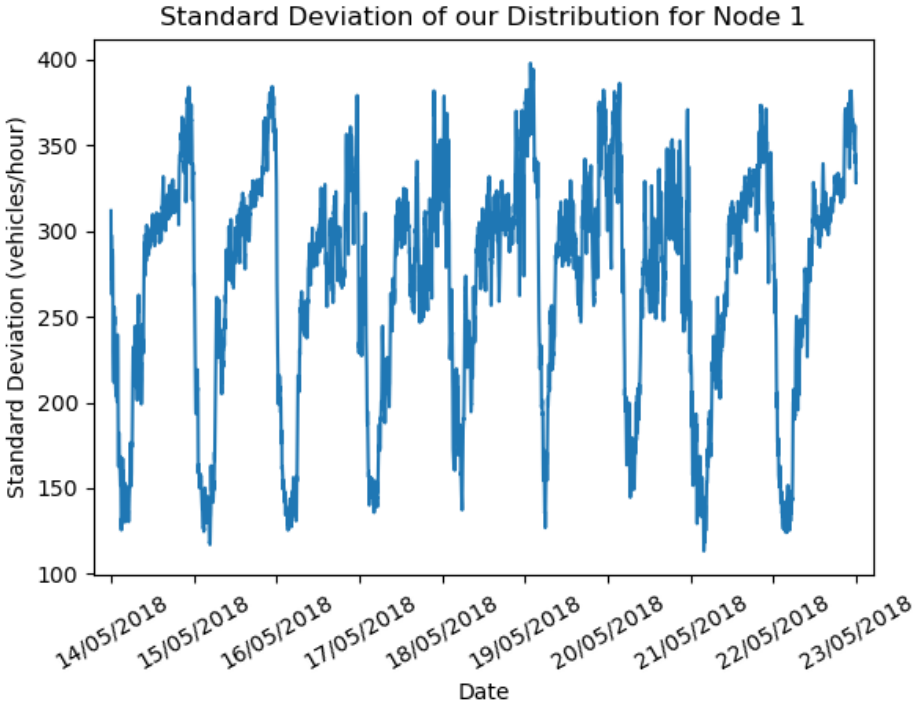


Figure C.13: Node 1

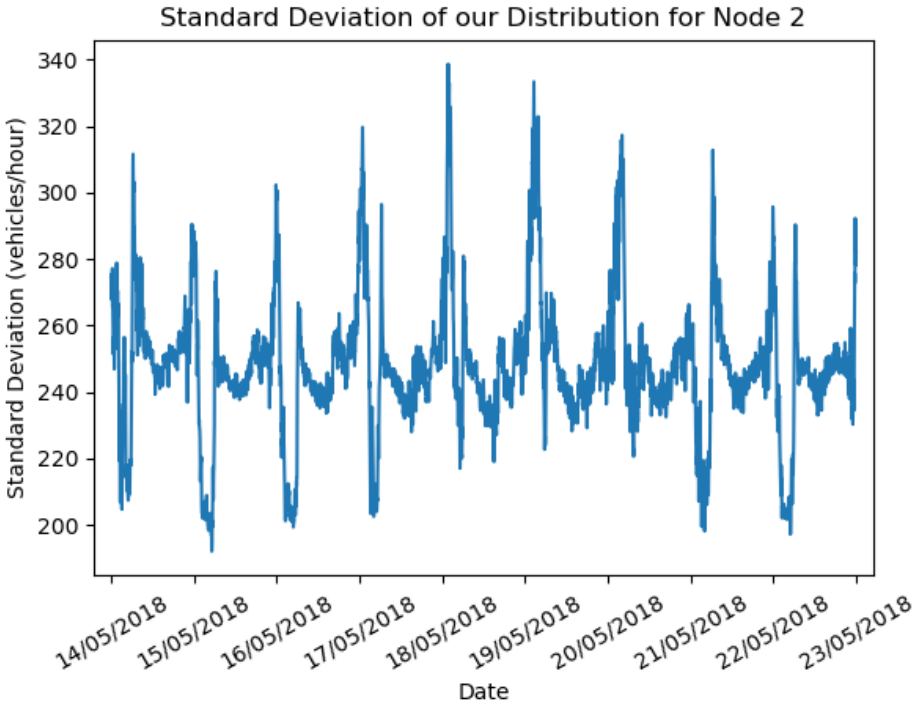


Figure C.14: Node 2

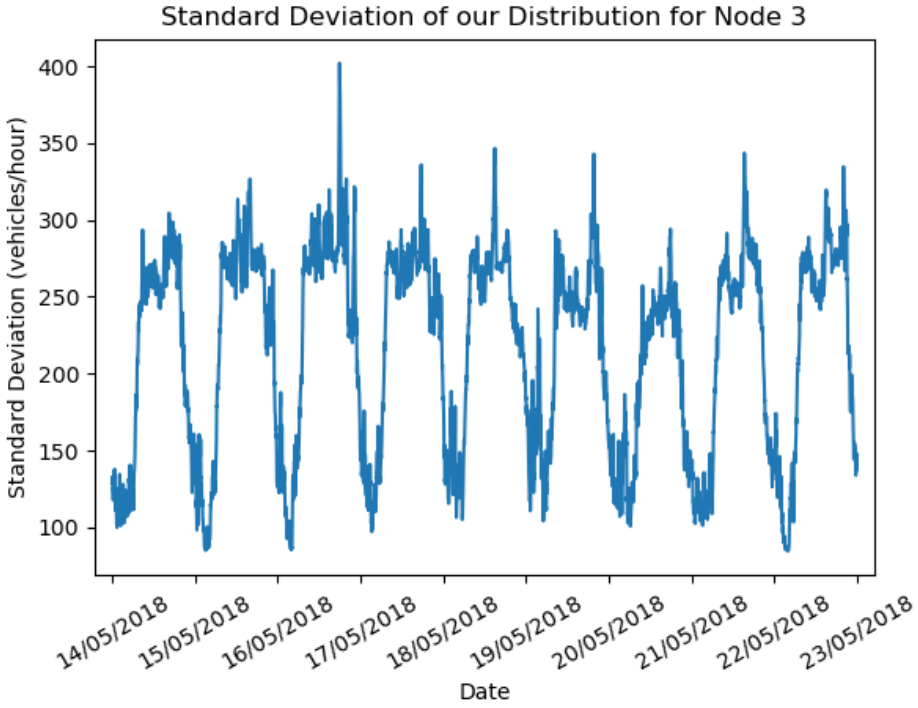


Figure C.15: Node 3

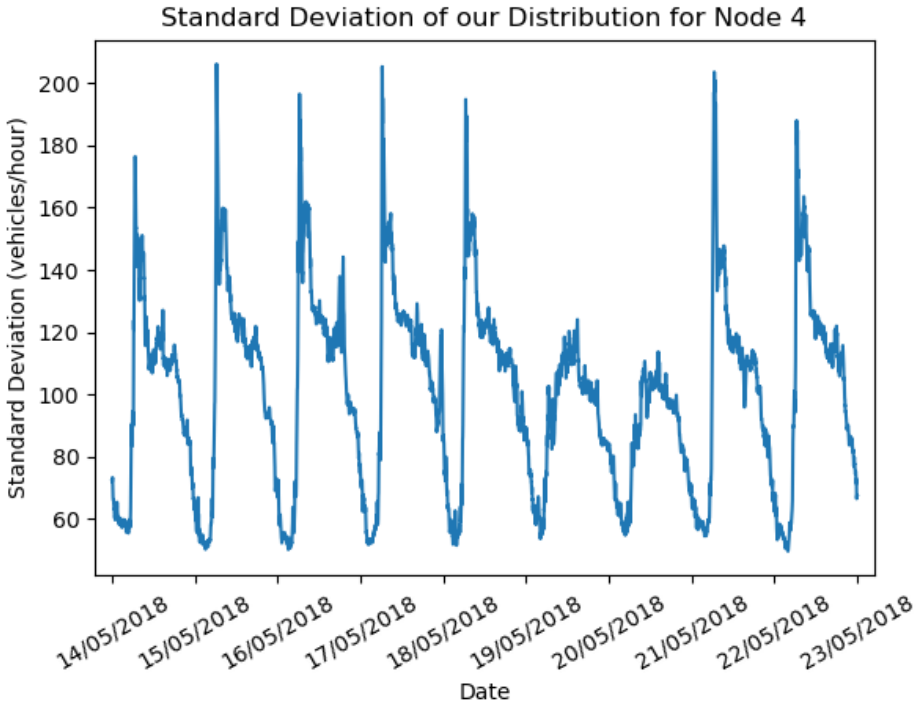


Figure C.16: Node 4

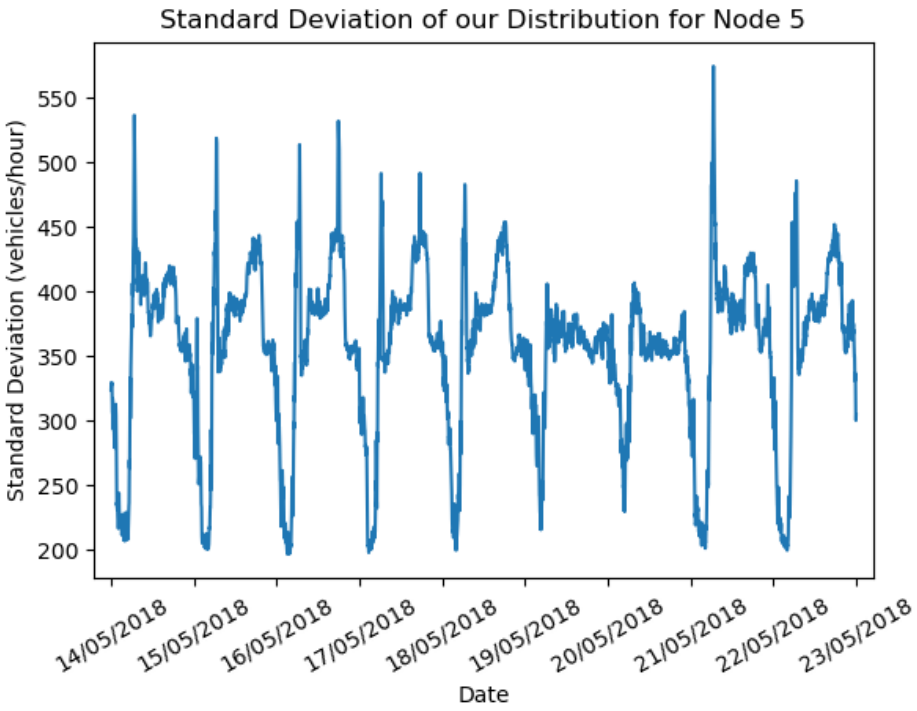


Figure C.17: Node 5

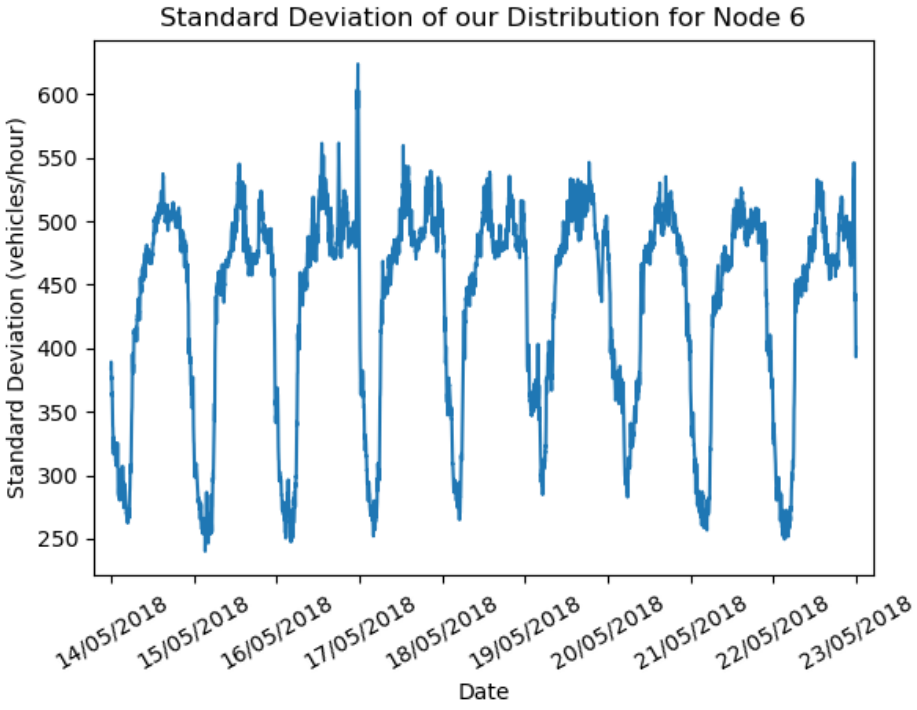


Figure C.18: Node 6

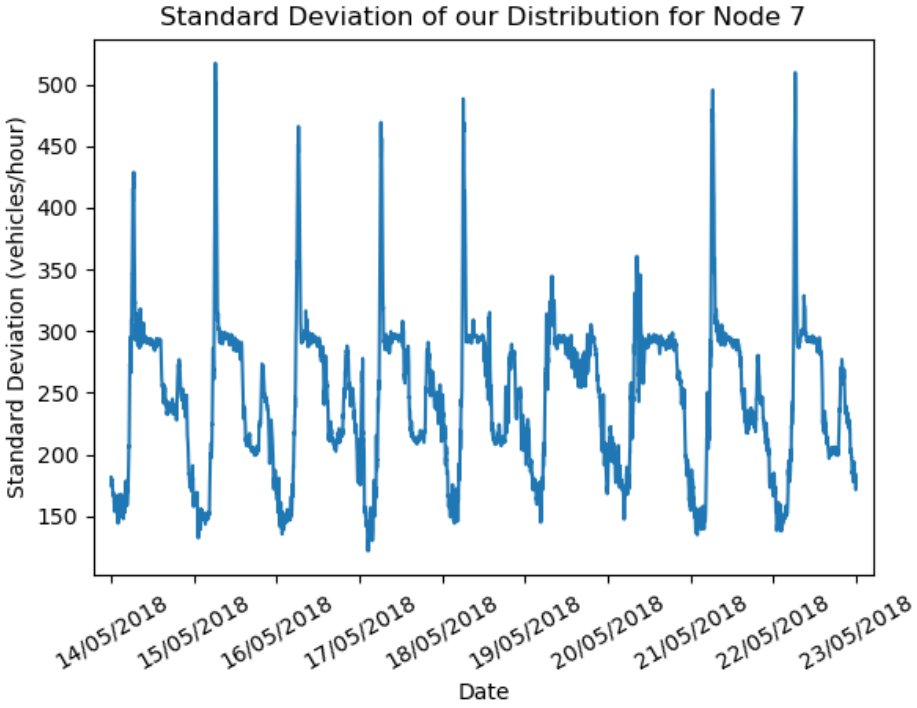


Figure C.19: Node 7

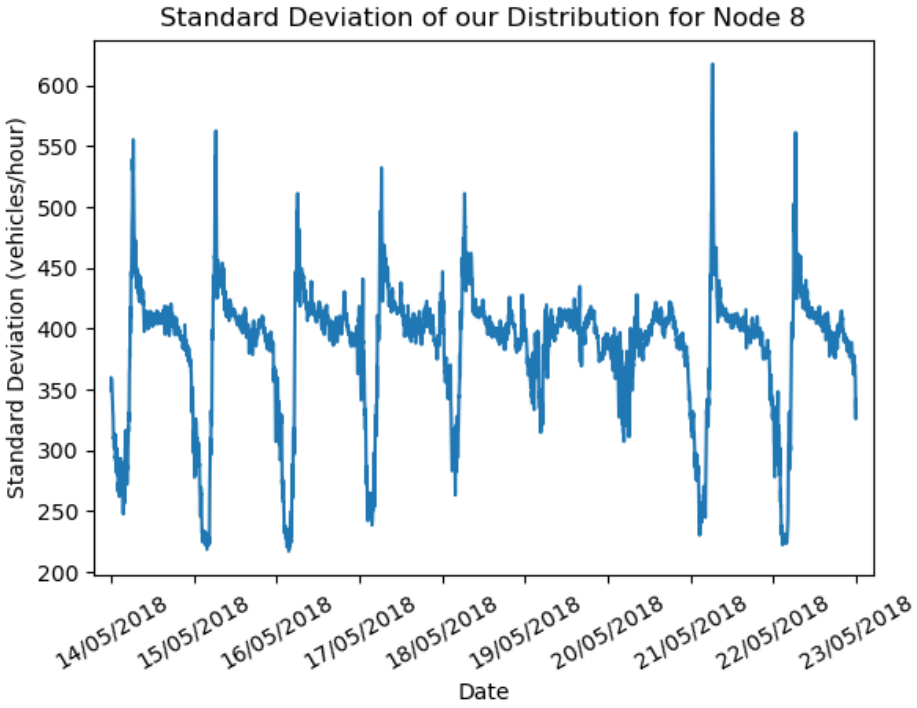


Figure C.20: Node 8

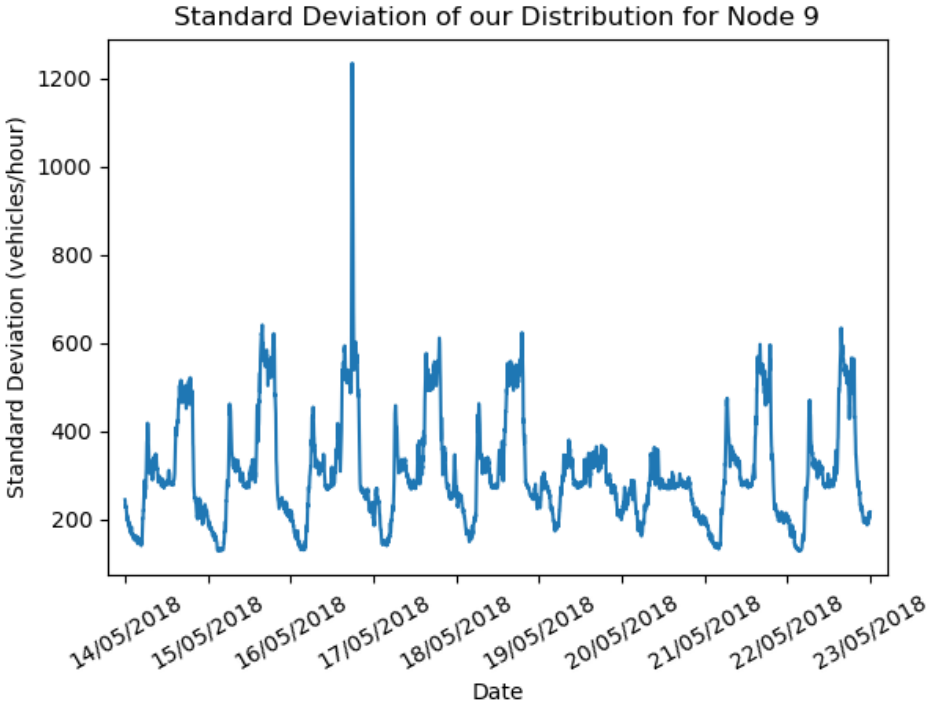


Figure C.21: Node 9

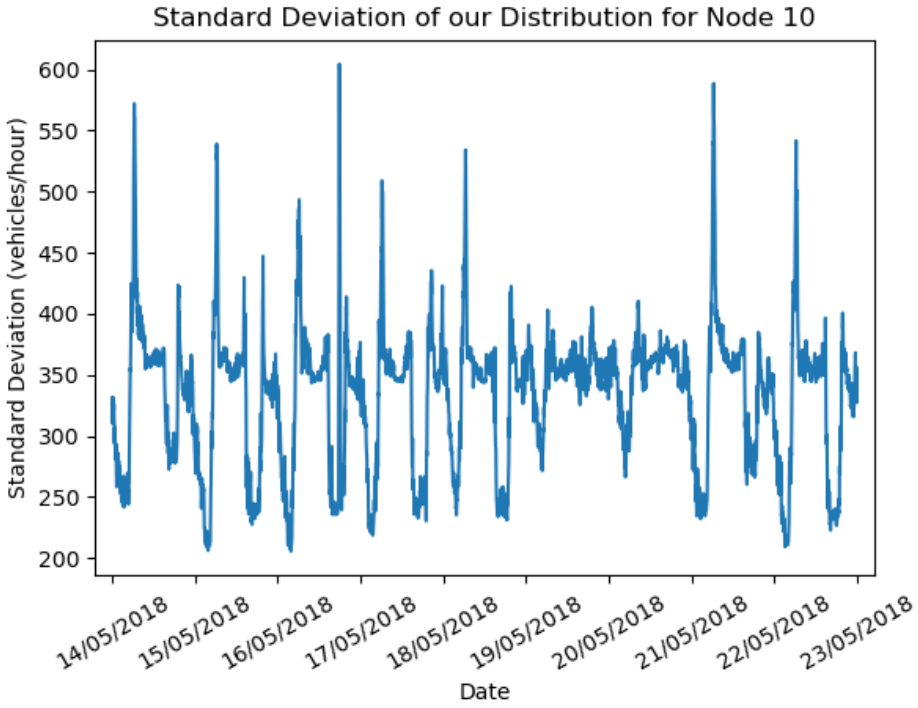


Figure C.22: Node 10