SHORT-TERM TRAFFIC DELAY PREDICTION AT THE NIAGARA FRONTIER BORDER CROSSINGS USING DEEP LEARNING

By

Rishabh Singh Chauhan

April 16, 2019

A thesis submitted to the

faculty of the Graduate School of

the University at Buffalo, The State University of New York

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Civil, Structural and Environmental Engineering

DEDICATION

To my parents Mrs. Archana Chauhan - Mr. Rajeev Singh Chauhan, and my sister Dr. Apoorva Chauhan.

ACKNOWLEDGMENT

This work would not have been possible without God's blessings and help.

I am extremely grateful to my advisor, Dr. Adel Sadek for his invaluable guidance, encouragement, and positivity towards my research throughout the Master's Program. It was a great learning experience. I would also like to sincerely thank Dr. Qing He for providing his feedback on the research design of this work and the directions for future research.

I am grateful to Dr. Lei Lin for guiding me to the online resources useful for learning the skill set required for this project. I would like to acknowledge Yunpeng Shi for all the research-related discussions. Furthermore, I would also like to thank Andrew Bartlett for sharing the data necessary for this project.

I am also thankful to my family. I am immensely grateful to my parents for all their support, motivation, and best wishes. Special thanks to my uncle Dr. Sanjay Chauhan, aunt Mrs. Sandhya Chauhan, brother Shashank Chauhan, and sister Apoorva Chauhan for all their help and guidance towards the completion of this project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Traffic delays at the United States-Canada border crossings have adverse effects on the economy as well as the environment. This thesis aims to predict passenger cars' traffic delays at the three Niagara Frontier Border Crossings, namely the Peace Bridge, the Lewiston-Queenston Bridge, and the Rainbow Bridge for the next 60 minutes into the future, using border wait time data collected by Bluetooth readers recently installed at the crossings. Firstly, the delay data were analyzed to assess the influence of bridges, weekdays and weekends, months/seasons, holidays, and direction of travel on traffic delay. After which, traffic delays were predicted using four deep learning techniques: Multilayer Perceptron (MLP), Convolutional Neural Networks (CNN), Long Short-Term Memory Recurrent Neural Networks (LSTM-RNN), and Gated Recurrent Unit Recurrent Neural Networks (GRU-RNN). The prediction accuracies of these models were evaluated by computing the Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared. The study also compared the performance of models trained on the whole data set (called 'complete set' models), versus models that were developed separately for weekdays and for weekends, to examine the effect of data classification on the models' predictive accuracy. The results suggest high-level accuracy of the deep learning techniques in predicting future traffic delays at the border crossings, with MAEs less than 3.5 minutes in predicting delays for up to 60 minutes into the future. However, no one deep learning technique emerged as a clear winner among others in predicting the delays. Findings of this thesis can guide future travelers in choosing the border crossing with the least delay, which in turn, should help in increasing the efficiency of the borders.

CHAPTER 1

INTRODUCTION

1.1 Background

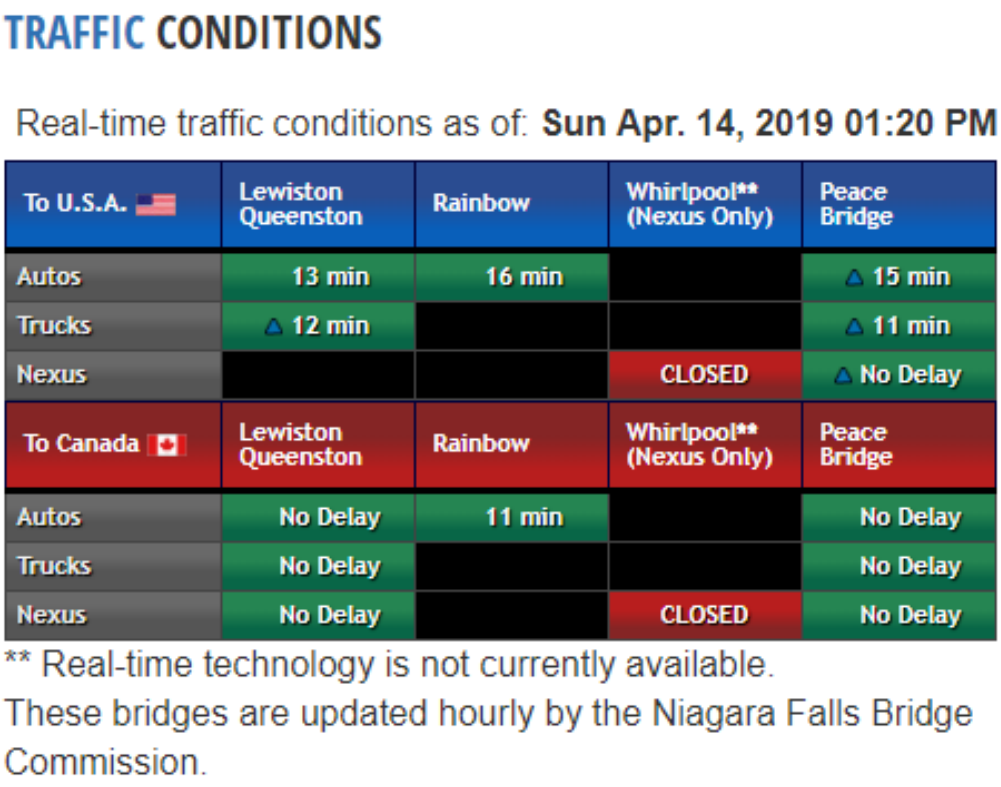
The United States and Canada share the world's longest border, which is a total of 8,891 kilometers stretching over land and water, and comprising of 119 border crossings ("The Canada-U.S. border: by the numbers | CBC News," 2011). Various analyses indicate the strong trade relations and high mobility between the two countries, like in 2010, the total Canada-U.S. trade summed up to $501 billion CDN with over $1 billion CDN trade daily ("The Canada-U.S. border: by the numbers | CBC News," 2011). Furthermore, a total of 140,728 number of cars and other vehicles crossed the border daily in the same year ("The Canada-U.S. border: by the numbers | CBC News," 2011). Additionally, it was found that approximately 300,000 people cross the U.S.-Canada border each day ("The Canada-U.S. border: by the numbers | CBC News," 2011).

Stephenson (Stephenson, 2016) has stated that for the past many years the U.S.-Canada border has remained unchanged physically, but the mental abstracts of the border have changed notably. During the end of the 20th century, the border was seen by many people as a formality between friends (Stephenson, 2016). Contributions were made by the Free Trade Agreement of 1987, the North American Free Trade Agreement (NAFTA) of 1994, and the Shared Border Accord of 1995 in improving the movement and reducing delay at the border (Stephenson, 2016). However, the major turning point in the history of mobility over the North American border came with the events of 9/11, after which the border wait time increased for both sides of the border due to the new screening processes (Stephenson, 2016).

The loss due to traffic delays at the border crossings has been estimated in various studies. The Ontario Chamber of Commerce found that delays at the U.S.-Canada border cost about $10 billion a year to the economies of both countries ("Border delays costing billions," 2004). As per a press release by the then U.S. Transportation Secretary Mary E. Peters, the delay experienced by the U.S. bound traffic from Canada costed more than $14 billion to the businesses in 2007 ("Press Release: US Department of Transportation Unveils New Program to Fight Border Congestion, 6/2/08 | Press Releases | Federal Highway Administration," 2008). A study conducted by Gabler and Moens (Gabler & Moens, 2012) estimated the annual cost of the border for Canada equaling to C$19.1 billion in 2010, which was found by adding the lowest value of the estimated cost ranges of trade, tourism, and government programs. This estimated annual cost was close to 1.5% of Canada's GDP in that year (Gabler & Moens, 2012). It has been estimated by the Canadian business group that by 2030, a direct cost of $17.8 billion in a year to both countries and a loss of 70,000 jobs in Canada can take place just because of the delays in the Detroit-Windsor corridor ("Border delays costing billions," 2004). Border delays and associated idling of vehicles waiting for inspection at the border has an environmental cost which comprises the cost of wasted fuel during idling, pollutants related to traffic, and health hazards (Lin, Wang, Sadek, & Li, 2014).

To enhance the security and accelerate the legitimate flow of people, goods, and services, the Prime Minister of Canada and the President of the United States issued 'Beyond the Border: A Shared Vision for Perimeter Security and Economic Competitiveness' in 2011 (Harper, United States. White House Office, & United States. President (2009- : Obama), 2011). In December 2011, an action plan was released to implement this shared vision, and a key commitment made in this was to implement a border wait-time measurement system at some Canada-United States border crossings, which were mutually determined as high priority (Transport Canada & U.S.

Department of Transportation – Federal Highway Administration, 2015). The top 20 high-priority borders, that included the Peace Bridge, the Lewiston-Queenston Bridge, and the Rainbow Bridge, were selected to implement these systems (Transport Canada & U.S. Department of Transportation – Federal Highway Administration, 2015). Bluetooth readers were installed at these three border crossings in recent years. The Bluetooth wait systems placed at the Peace Bridge and Lewiston-Queenston Bridge calculates average wait time by vehicle type and direction using Traffax readers and BluFaxWeb software (Roelofs, Preisen, & Helgeson, 2016).



TRAFFIC CONDITIONS

Real-time traffic conditions as of: **Sun Apr. 14, 2019 01:20 PM**

To U.S.A.	Lewiston Queenston	Rainbow	Whirlpool** (Nexus Only)	Peace Bridge
Autos	13 min	16 min		△ 15 min
Trucks	△ 12 min			△ 11 min
Nexus			CLOSED	△ No Delay
To Canada	**Lewiston Queenston**	**Rainbow**	**Whirlpool** (Nexus Only)**	**Peace Bridge**
Autos	No Delay	11 min		No Delay
Trucks	No Delay			No Delay
Nexus	No Delay		CLOSED	No Delay

** Real-time technology is not currently available.
These bridges are updated hourly by the Niagara Falls Bridge Commission.

Figure 1. 1: Traffic conditions at the Border Crossings as displayed on the Niagara Falls Bridge Commission Website ("Niagara Falls Bridge Commission," n.d.).

Real-time traffic conditions at the Peace Bridge, Lewiston-Queenston Bridge, and Rainbow Bridge are displayed on the websites like Niagara Falls Bridge Commission website and NITTEC website

("Niagara Falls Bridge Commission," n.d.; "NITTEC – Travel Smart," n.d.; Roelofs et al., 2016). Figure 1.1 shows the traffic conditions at the border crossings as displayed on the Niagara Falls Bridge Commission website ("Niagara Falls Bridge Commission," n.d.). NITTEC or Niagara International Transportation Technology Coalition is a coalition of fourteen different agencies in Southern Ontario and Western New York which provides the information about the current border crossing delays (Lin, Wang, Sadek, et al., 2014).

1.2 Niagara Frontier Border Crossings

The most frequently used border crossings between New York State and Ontario are in the Niagara Falls area at Peace Bridge, Lewiston-Queenston Bridge, Rainbow Bridge, and Whirlpool Bridge crossings (Roelofs et al., 2016). As the Whirlpool Bridge is a NEXUS only border crossing (Roelofs et al., 2016), in this study, the focus has been on the other three border crossings. This section provides information about these three border crossings. Figure 1.2 shows the location of these border crossings (Zhang & Lin, 2017).

The Peace Bridge is located between Buffalo, New York, U.S. and Fort Erie, Ontario, Canada (Roelofs et al., 2016). The Buffalo and Port Erie Public Bridge Authority own and operate this bridge (Roelofs et al., 2016). The Peace Bridge includes 20 inspection lanes on the Canadian side and 18 primary inspection lanes on the United States side (Roelofs et al., 2016). FAST and NEXUS lanes are provided on it (Roelofs et al., 2016).

The Lewiston-Queenston Bridge connects Lewiston, New York, U.S. and Queenston, Ontario, Canada (Roelofs et al., 2016). The Niagara Falls Bridge Commission own and operate this bridge (Roelofs et al., 2016). It comprises of 15 primary inspection lanes for the Canada bound traffic and 10 primary inspection lanes for the U.S. bound traffic (Roelofs et al., 2016). While the FAST lanes

are available at both sides of the crossings, NEXUS lanes are only on the Candian side (Roelofs et al., 2016).

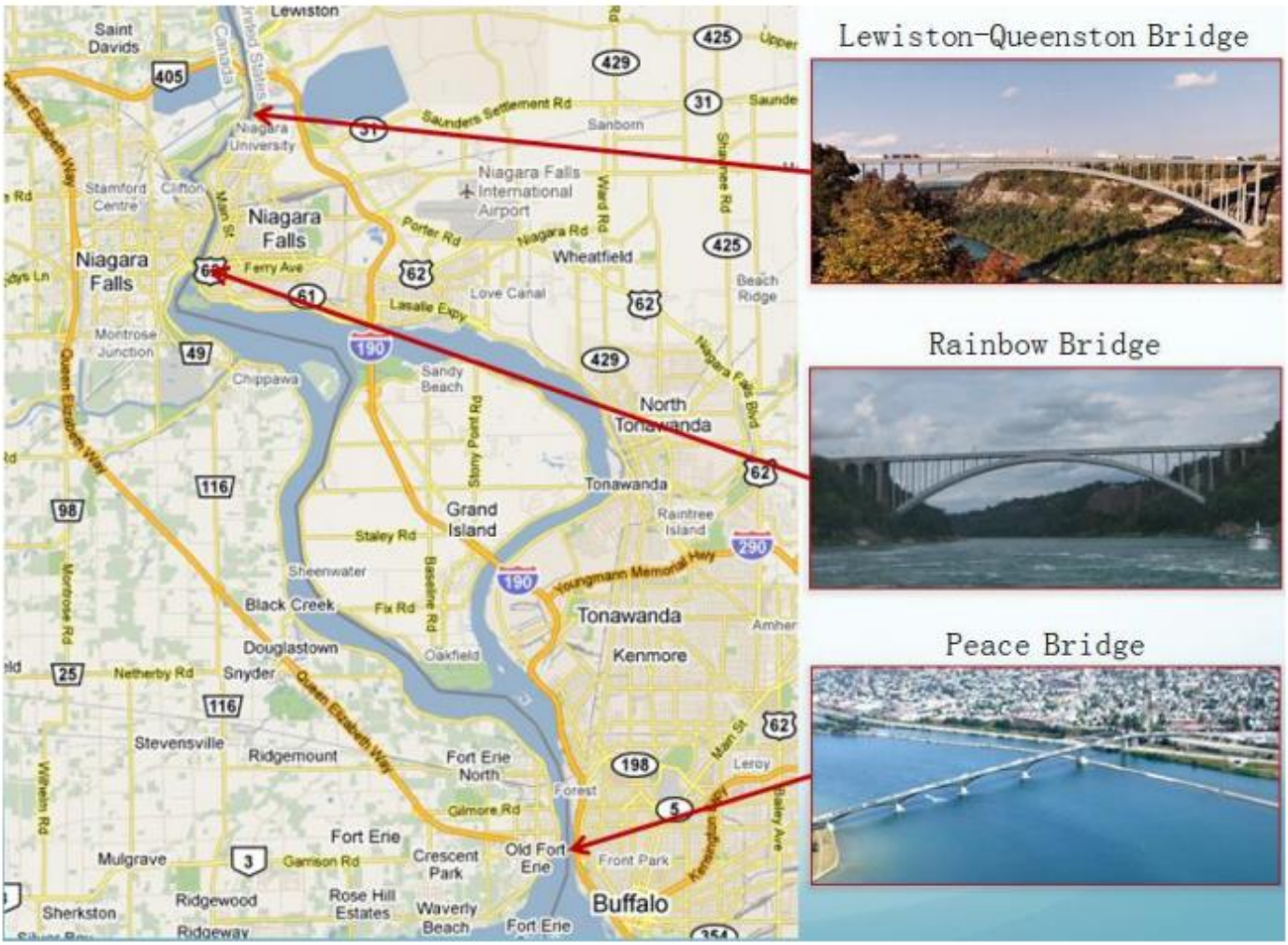


Figure 1. 2: Locations of the Lewiston-Queenston Bridge, Rainbow Bridge and Peace Bridge (Zhang & Lin, 2017)

The Rainbow Bridge connects Niagara Falls, New York, U.S. to Niagara Falls, Ontario, Canada (Roelofs et al., 2016). The Niagara Falls Bridge Commission own and operate this bridge (Roelofs et al., 2016). 17 primary inspection lanes are provided for the Canada bound traffic, while 15 primary inspection lanes are provided for the U.S. bound traffic (Roelofs et al., 2016). A dedicated

Ready lane for RFID (Radio-frequency identification) -enabled devices and NEXUS lanes for traffic going in both directions are available on the Peace Bridge (Roelofs et al., 2016). Moreover, commercial traffic is not allowed on it (Roelofs et al., 2016).

1.3 Purpose and Scope

1.3.1 Major Research Goals

As stated by Lin et. al. (Lin, Wang, Sadek, et al., 2014), the future wait time that will be experienced by the traveler when they arrive at the border might differ from the current wait time and hence, predicting the future border crossing delay is necessary. The predicted future border crossing delay can be helpful for border crossing authorities in determining the needed staffing level and also, in routing the border-destined traffic intelligently (Lin, Wang, Sadek, et al., 2014). This was one of the motivations for this study.

The major research goal is to accurately predict the traffic delay experienced by the U.S. bound passenger cars up to next one hour by using the current delay data available from the Bluetooth readers placed recently at the three border crossings.

In this study, the border crossing delay refers to the difference in the actual travel time and the travel time in base conditions across the two points measured by Bluetooth readers.

1.3.2 Focus of the Current Study

There are several research areas that are focused on the current study. This study focuses on achieving the following goals:

1.3.2.1 Analyzing the Border Crossing Delay

The delay data from the three Niagara Frontier Border Crossings were analyzed to assess the influence of bridges, weekdays and weekends, months/seasons, holidays, and direction of travel on traffic delay.

1.3.2.2 Short-term Traffic Delay Prediction

The study aims to predict the traffic delay for the next 5 minutes, 10 minutes, 15 minutes, 20 minutes, and so on up to the next 60 minutes into the future. To my best knowledge, this is the first study in which the traffic delay is predicted for the three Niagara Frontier Border Crossings: Peace Bridge, Lewiston-Queenston Bridge, and Rainbow Bridge.

1.3.2.3 Development and Comparison of Different Deep Learning Methods

The traffic delays were predicted using four deep learning techniques, namely Multilayer Perceptron (MLP), Convolutional Neural Networks (CNN), Long Short-Term Memory Recurrent Neural Networks (LSTM-RNN), and Gated Recurrent Unit Recurrent Neural Networks (GRU-RNN). The prediction accuracies of these models were evaluated and compared by computing the Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared. The study also compared the performance of models trained on the whole data set (called 'complete set' models), versus models that were developed separately for weekdays and for weekends, to examine the effect of data classification on the models' predictive accuracy.

1.3.2.4 Traffic Management at the Border Crossings

It was aimed to predict the traffic delay at the border crossings into the future. This information, if provided in real-time to the travelers can guide them. Travelers will most likely have the tendency

to choose the border crossing with the least delay. This would help in the uniform distribution of traffic across the crossings. Hence, Findings of this thesis can guide future travelers in choosing the border crossing with the least delay, which in turn, should help in increasing the efficiency of the borders.

1.4 Thesis Overview

This thesis consists of five chapters. Followed by this introduction chapter, is the literature survey chapter which presents the previous studies related to the current study. Chapter 3 contains a description of the methodology employed in this research. Chapter 4 shows the result of the research and discusses them. Lastly, Chapter 5 concludes the findings of this study and presents the suggestions for future work.

CHAPTER 2

LITERATURE REVIEW

The study presented in this thesis involves the analysis of delays at the border crossings and the delay prediction by deep learning. In the past, there have been several studies related to these topics. This chapter presents some of these notable past studies.

2.1 Border Crossing Delay Prediction and Analysis

There have been numerous studies in the past that focused on delays at the border crossings (A. M. Khan, 2010; Lin F. B. & Lin M. W., 2001; Lin, Wang, & Sadek, 2014; Lin, Wang, Sadek, et al., 2014; Moniruzzaman, Maoh, & Anderson, 2016; Zhang & Lin, 2017; Zhang, Lin, Zhu, & Sharma, 2017). These studies are discussed in this section.

2.1.1 Analyzing Passenger Cars' Traffic Delay at the Niagara Frontier Border Crossings

Researchers have been interested in analyzing the traffic delays at the Niagara Frontier border crossings (Zhang & Lin, 2017; Zhang et al., 2017). Zhang et. al. (Zhang et al., 2017) aimed to identify the various delay patterns over the Peace Bridge, the Lewiston-Queenston Bridge, and the Rainbow Bridge by building a decision tree for each vehicle type (passenger and commercial vehicles) for each direction of travel (the U.S. bound and Canada bound). A decision tree is a modern machine learning model and a decision support tool that uses a flowchart-like structure (Zhang et al., 2017). The wait time dataset used in this study comprised of the wait times for both vehicle types, moving in both directions over the three bridges between 7:00 to 21:00 (Zhang et al., 2017). As the wait times from the Rainbow Bridge updates hourly due to the unavailability of

Bluetooth technology, the five-minute wait times from the other two bridges were also aggregated to hourly wait times, so as to make the data from each bridge comparable (Zhang et al., 2017). By analyzing the frequencies of delay patterns across the three bridges for each vehicle type in both directions, it was found that the dominant patterns show the delay status is uneven across these bridges (Zhang et al., 2017). Hence, it was concluded that the border crossing delays for the same vehicle types in the same direction are not evenly distributed over the three Niagara Frontier Bridges (Zhang et al., 2017). It was also found that due to the fact that the Peace Bridge and the Lewiston-Queenston Bridge connects the Canadian freeways with USA I-90, these two bridges might be congested for the U.S. bound passenger vehicles especially when there is a holiday in Canada (Zhang et al., 2017).

Zhang and Lin (Zhang & Lin, 2017) processed the historical Niagara Frontier border wait times data by applying a dictionary-based compression algorithm to identify the abnormal spatial-temporal patterns for both passengers vehicles and trucks at the three border crossings – the Peace Bridge, the Lewiston-Queenston Bridge, and the Rainbow Bridge. The abnormal patterns were ranked and analyzed (Zhang & Lin, 2017). The dataset used in this study comprised of wait times experienced by both types of vehicles, moving either U.S. bound or Canada bound, over the three bridges (Zhang & Lin, 2017). Zhang and Lin (Zhang & Lin, 2017) have stated that the wait times at the Peace Bridge and the Lewiston-Queenston Bridge updated in every 5 minutes but it updated hourly for the Rainbow Bridge due to the unavailability of Bluetooth technology. Hence, the 5 minutes wait times from the Peace Bridge and the Lewiston-Queenston Bridge were replaced with hourly wait times by taking the average for each hour, so as to eliminate the effects of fluctuations and match the wait times obtained from all three bridges (Zhang & Lin, 2017). By analyzing the top three abnormal patterns for different vehicles and directions, it was found that the weekends

and holidays can lead to unusual heave congestion on the three bridges at the same time (Zhang & Lin, 2017). Additionally, it was found by correlating the time-of-day information with the top 5% abnormal patterns that the frequency of abnormal wait time patterns for U.S. bound cars and trucks reaches to the peak during the afternoon (Zhang & Lin, 2017).

2.1.2 Border Crossing Traffic Delay Prediction

There have been several studies in the past that aims to estimate the traffic delays at the border crossings (A. M. Khan, 2010; Lin F. B. & Lin M. W., 2001; Lin, Wang, & Sadek, 2014; Lin, Wang, Sadek, et al., 2014; Moniruzzaman et al., 2016; Zhang & Lin, 2017; Zhang et al., 2017). Lin and Lin (Lin F. B. & Lin M. W., 2001) identified the characteristics of vehicle-processing time at the three border crossings: the Thousand Islands, Ogdensburg, and Seaway crossings. Additionally, they also proposed a delay model. As the collection of enough field data for finding the relationships between the delay and its numerous governing factors was found to be very difficult, simulated delay data was used in this study (Lin F. B. & Lin M. W., 2001). The delay model proposed by Lin and Lin (Lin F. B. & Lin M. W., 2001) was based on various factors like vehicle processing capacity of a toll gate or inspection gate, volume/capacity ratio, number of available gates, etc. It was found that the estimates of average approach delays gotten from this delay model rarely deviate by more than 10% from the simulated values, when the volume/capacity ratio was below 1.4 (Lin F. B. & Lin M. W., 2001).

Khan (A. M. Khan, 2010) has predicted the delays at the Ambassador bridge crossing linking Windsor in Ontario, Canada, and Detroit in Michigan, U.S. In this study, a model of border crossing system including the access road was prepared in VISSIM microsimulation software which was then calibrated and validated (A. M. Khan, 2010). Further, Artificial Neural Networks

(ANN) models were developed to forecast queues and delay on the bridge (A. M. Khan, 2010). The ANN models were trained using the inputs and outputs of the calibrated microsimulation model and the data for the development of ANN models were obtained from the repeated runs of the stochastic microsimulator (A. M. Khan, 2010). A two-layer neural network model with a feed-forward structure was selected as the structure of the ANN model (A. M. Khan, 2010). The delay estimated by VISSIM and the ANN model were compared and it was found to have a high correlation with R^2 over 0.97, which suggest a highly satisfactory model (A. M. Khan, 2010).

Another study conducted by Lin et. al. (Lin, Wang, & Sadek, 2014) developed a multi-server queueing model to predict the border crossing traffic delay at the Peace Bridge. Lin et. al. (Lin, Wang, & Sadek, 2014) suggested a two-step solution for the problem of providing predictive information about border crossing delay. The first step is the short-term traffic volume prediction and the second step involves formulating and solving the appropriate queueing model for predicting the expected delay as a function of the estimated arrival volume obtained from the first step (Lin, Wang, & Sadek, 2014). This study focuses on the second step of the solution (Lin, Wang, & Sadek, 2014). In this study, the border crossing delay was predicted by two groups of multi-server queueing models, namely M/ E_K/ n and BMAP/PH/n queueing models which were developed based on the real-time traffic volume and inspection time data collected from the Peace Bridge (Lin, Wang, & Sadek, 2014). The transient solutions of these models were derived using heuristic approaches and were validated by comparing the model results with those obtained from a microscopic traffic simulation model (Lin, Wang, & Sadek, 2014). By conducting sensitivity analysis, it was concluded that the solutions of the queueing model are reasonable (Lin, Wang, & Sadek, 2014).

Some other efforts made to predict traffic delay at the border crossing include the study conducted by Lin et. al. (Lin, Wang, Sadek, et al., 2014). This study introduced an Android smartphone application which not just provides the historical and current waiting times, but also wait times predicted for the next 15 minutes at the Niagara Frontier border crossings (Lin, Wang, Sadek, et al., 2014). They predicted the future delay by using the stepwise border crossing delay prediction model described by Lin et. al. (Lin, Wang, & Sadek, 2014), which involves first the short-term prediction of traffic volumes arriving at the border crossings and then solving a transient multi-server queueing problem to predict the delay using the results obtained from the first step as input (Lin, Wang, Sadek, et al., 2014). The border crossing traffic volumes were predicted by Seasonal Autoregressive Integrated Moving Average (SARIMA) due to its moderate computational cost and easiness of implementation (Lin, Wang, Sadek, et al., 2014). The $M / E_K / n$ queueing model was used for predicting the delay (Lin, Wang, Sadek, et al., 2014). The dataset used in this study was the traffic volume data which was used as input to develop the stepwise border delay prediction model, and also the current waiting times (Lin, Wang, Sadek, et al., 2014). When the model predictions were compared with the real-world delay, the model was found to have a mean absolute difference of about 6.6 minutes (Lin, Wang, Sadek, et al., 2014).

Recently, Moniruzzaman et. al. (Moniruzzaman et al., 2016) developed Multilayer feedforward ANN with backpropagation approach were used to forecast the crossing times for trucks at the Ambassador Bridge border crossing. Two sets of ANN models were developed for the short-term prediction of the volumes and crossing times for trucks at the border crossing (Moniruzzaman et al., 2016). The ANNs were designed, trained and validated using a volume data obtained from Remote Traffic Microwave sensors and Global Positioning System data for crossing time (Moniruzzaman et al., 2016). The training of ANN model for predicting the truck volume relied

only on the time lags of the dependent variable, whereas that for predicting the crossing time relied on lags of crossing time, truck volume on the bridge, hours of day, and day of week (Moniruzzaman et al., 2016). ARIMAX (ARIMA with Exogenous Inputs), multilayer perceptron, and radial basis function network were also estimated to compare their prediction accuracy with the crossing time ANN model developed in this study and it was found that this ANN model had the lowest mean absolute percentage error (MAPE) among the other modeling approaches in both U.S. bound and Canada bound directions (Moniruzzaman et al., 2016).

2.2 Time Series Prediction

As per Masum et. al. (Masum, Liu, & Chiverton, 2018), time series (X) is given by:

$$X = (x_t; t = 1, 2, \ldots, N)$$

Where, x_t is a measurement at an individual time point t (Masum et al., 2018).

Time series prediction or forecasting has found its application in numerous tasks, for instance, electric load prediction (Masum et al., 2018), solar power forecasting (Koprinska, Wu, & Wang, 2018), traffic speed prediction (Ma, Tao, Wang, Yu, & Wang, 2015), traffic flow prediction (Fu, Zhang, & Li, 2016; Polson & Sokolov, 2017), telephonic activity load forecasting (Bianchi, Maiorino, Kampffmeyer, Rizzi, & Jenssen, 2017) and wind forecasting (Dalto, Matuško, & Vašak, 2015). There are numerous methods suggested in the literature for time series prediction. However, in recent years, many researchers have been interested in understanding the potential of deep learning methods in the prediction of time series. This section presents several studies that were conducted to forecast time series.

Researchers have been interested in comparing various time series prediction methods. Masum (Masum et al., 2018) conducted and compared the multi-step forecasting of univariate time series by Auto-Regressive Integrated Moving Average (ARIMA) and Long-Short-Term-Memory (LSTM) based Recurrent Neural Networks (RNN) models. The forecasting was performed for three nonlinear electric load datasets with their frequency downsampled to 1 day. For formulating ARIMA model, they found through ADF Statistic that d parameter should be at least 1 for better performance. Further, they computed, plotted and examined the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) for identifying the parameters p and q of the ARIMA model. For formulating the LSTM model, they applied seasonal differencing to the data for transforming it to stationary time series data and then rescaled it to be between -1 and 1. They developed the LSTM model having 1 hidden layer with 1 LSTM unit, Root Mean Squared Error (RMSE) as loss function, ADAM algorithm as an optimizer, epoch and batch size was set to 1. Finally, they divided each dataset into train and test, and forecasted the time series using both the methods for next 1, 2, 3, 4, and 10 days. The comparison of forecasting results from the two methods using RMSE suggested that the LSTM model performed better than the ARIMA model in multi-step forecasting of the time series (Masum et al., 2018).

Some researchers have compared the performance of deep learning techniques in time series prediction with shallow neural networks. Dalto et. al. (Dalto et al., 2015) developed a deep neural network model, specifically a feed-forward multilayer perceptron (MLP) for ultra-short-term wind prediction at three different locations. The model comprised of 4 layers, hyperbolic tangent functions as activation functions, mean squared error as loss function and L2 regularization. It was developed using wind datasets collected from specific locations, which was normalized to be between -1 and 1, and each was split into training sampling (70%), validation (15%) and testing

(15%). The predictions were made for three-hour time horizon with 10 minutes resolution in north and east direction. The deep learning method was compared using mean absolute error (MAE) with a persistence prediction and a shallow neural network (SNN). The persistent prediction assumed a constant wind speed and direction for the prediction horizon. Based on the comparison of prediction performance, the authors have concluded that the deep neural networks give better results than shallow neural networks and persistent forecast and hence, they are better than the other two methods for ultra-short-term wind prediction (Dalto et al., 2015).

A similar type of results that suggest the superiority of deep learning approaches in time series prediction was obtained by some other studies also, like Polson and Sokolov (Polson & Sokolov, 2017), who developed an innovative deep learning architecture for time series forecasting to predict the short-term traffic. This architecture comprised of a linear model fitted by l_1 regularization, along with a sequence of tanh layers. They compared the performance of deep learning model with sparse linear vector autoregressive (VAR), which involved several data pre-filtering techniques, specifically, median filtering with a window size of 8 measurements and trend filtering with λ=15, by computing mean squared error (MSE) and R^2. Further, they compared deep learning with compared a simple neural network model having a single hidden layer. They applied their methodology to predict traffic flows during a Chicago Bears football game and an extreme snowstorm event, using road sensor data collected from the interstate. The authors concluded from the result of this study that the short-term traffic forecasting by deep learning models is clearly improved than that by linear models and one-layer network model, and also, that some other types of networks like recurrent neural networks (RNNs) and long short-term memory (LSTM) have given superior performance for time series data. However, they pointed out that the problem with deep learning models is low explanatory power (Polson & Sokolov, 2017).

In some studies, the time series prediction performance of a specific architecture of neural networks is compared with other architectures. One such study was conducted by Ma et. al. (Ma et al., 2015) in which the Long Short-Term Memory Neural Networks (LSTM) were developed to forecast time series for predicting the traffic speed in next 2 minutes and were compared with Autoregressive Integrated Moving Average (ARIMA), Support Vector Regression Machine (SVM) Kalman Filter approaches, Regression and other Recurrent Neural Network (RNN) structures, namely, Time-delayed NN, Elman NN, and Nonlinear Autoregressive NN. The traffic speed data that was used in this study was collected by microwave traffic detectors from two separate locations at an expressway without signal controls. The data was split into two groups: first 25 days for training and the last 5 days for testing the model. The LSTM models were developed for different time lags and input combinations, i.e. only speed or speed and volume. The comparative analysis showed that LSTM performed the best in terms of accuracy and stability, out of all the other algorithms tested. The authors have concluded that the features of LSTM in time series prediction, namely, to find the optimal time lags automatically and to learn the time series with long time dependencies, are needed for traffic prediction problems (Ma et al., 2015).

Another study conducted by Fu et. al. (Fu et al., 2016) also showed that deep learning models like LSTM and Gated Recurrent Unit (GRU) neural network (NN), based on RNN outperformed the ARIMA in traffic flow prediction by time series forecasting. The models were developed to predict traffic flow for the next 5 minutes using the traffic flow data of the past 30 minutes. The data used in this study was collected from sensors and unique models were developed for each sensor. The models were trained using the data for three weeks and the testing was conducted on the data of the following week. The prediction accuracy of these models was compared for 50 randomly

chosen sensors, using MSE and Mean Absolute Percentage Error (MAPE). The results showed that the GRU NN model performed the best and ARIMA model did worst (Fu et al., 2016).

Some studies focused on comparing the time series prediction of various deep learning methods like the study conducted by Bianchi et. al. (Bianchi et al., 2017), in which the comparison was made amongst the different architectures of RNNs, namely Elman Recurrent Neural Network (ERNN), Non-linear Auto-Regressive with eXogeneous inputs (NARX), Echo State Network (ESN), LSTM, and GRU. In this study, short term load forecast is made by 24 hours ahead prediction using three real-world time series, two of which included exogenous variables. Each time series was split into a training set, validation set, and testing set. The prediction accuracy was measured by ψ, which is equal to 1 – Normal Root Mean Squared Error (NRMSE). By conducting a comparative analysis of prediction performance of the five models, the authors concluded that in terms of performance and simplicity, ERNN and ESN might be the most convenient choice for time series prediction problems. However, they also pointed out that none of the five types of RNN architectures was able to provide the best results in all the prediction problems studied and so they suggested that the best choice among the RNN architecture depends on the task that has to be performed (Bianchi et al., 2017).

In recent years, some studies applied Convolutional Neural Networks (CNNs) to real-world time series prediction and compared their performance with other popular methods. One such study was conducted by Koprinska et. al. (Koprinska et al., 2018), in which the solar power and electricity load of previous days were used to predict the next day half hourly solar power between 7 am and 5 pm and the next day hourly electricity load respectively. This task was performed by using CNN, LSTM RNN, standard backpropagation MLP with a single hidden layer, and a persistence baseline model. The performance of these models was evaluated and compared through MAE and Root

Mean Square Error (RMSE) on four time series, one of which contained solar power data and other three had electricity load data. The solar data comprised of PhotoVoltic (PV) power data collected from a rooftop PV plant and aggregated to 30 minutes interval for this study. On the other hand, the electricity load dataset was obtained from three countries and was sampled every hour. These time series datasets contained data of 2 years, out of which, 70% of the first year was used as training set used for training the models, 30% of the second year was used as validation set used for selecting the parameters of the models and the data of the second year as testing set used for evaluating the prediction accuracy. The architecture and parameters of CNN were selected by choosing the combinations which gave the best performance on the validation set among all the other combinations tried. The persistence model was developed such that the forecast for the next day was the same as the input values of the previous day. By comparing the performance of these models on all four time series, the authors have concluded that the most accurate prediction methods are CNN and MLP, while the least accurate was LSTM. Additionally, the authors also noted that LSTM took a longer time to train than the CNN and MLP models (Koprinska et al., 2018).

Considering the results obtained from the above-mentioned studies, it seems that there is enough evidence in the literature that proves the superior performance of deep learning techniques in predicting time series. These deep learning techniques have been found better than linear models (Polson & Sokolov, 2017) and shallow neural networks (Dalto et al., 2015) in forecasting the time series. To sum up, the comparative analysis of time series prediction performance made in (Dalto et al., 2015), (Koprinska et al., 2018), (Ma et al., 2015), and (Fu et al., 2016) have shown that MLP, CNN and MLP, LSTM RNN, and GRU RNN respectively have outperformed the other methods used in that study. Taking into considering the experimental results gained from these

studies, MLP, CNN, LSTM RNN, and GRU RNN models were selected in this study for forecasting the traffic delay time series to predict the short-term traffic delay at the border crossings.

CHAPTER 3

METHODOLOGY

3.1 Modeling Dataset

The dataset used for modeling was obtained from the Niagara International Transportation Technology Coalition (NITTEC). It contains the traffic delay data for the U.S. bound car traffic moving over each of the three Niagara frontier border crossing bridges - Peace Bridge, Lewiston-Queenston Bridges or Queenston-Lewiston Bridge, and Rainbow Bridge. It was collected by Bluetooth readers at a frequency of one minute. For this study, the data used was collected from March 1, 2018, to December 31, 2018, which is a total of ten-months data.

The data had very few missing points, only 24 out of a total of 13,21,920 data points from all the three bridges combined. The missing values were filled by taking the average of the previous value and the next available value. By observing the data, it was found that the traffic delay did not fluctuate much every minute and hence, it seemed rational to aggregate the data to five-minute intervals. This U.S. bound traffic delay data of ten months on the three bridges aggregated to five-minute intervals is referred to as ‘complete set’.

Further, to evaluate the effect of days of the week on traffic delay and also, to assess the ability of deep learning techniques to model categorized data, the complete set was split into ‘weekday set’ and ‘weekend set’. The weekday set consisted of the delay data only during weekdays, whereas the weekend set comprised of delay data only during weekends. Therefore, three sets of data were

available for the study – complete set, weekday set, and weekend set. The complete set, weekday set, and weekend set comprise of 88128, 62784, and 25344 data points respectively.

3.2 Data Collection

The data used for this study was collected by the Bluetooth readers installed in recent years at the border crossings: The Peace Bridge, the Lewiston-Queenston Bridge, and Rainbow Bridge. This data had the information about the U.S. bound passenger cars' traffic delay collected at an interval one minute. The Bluetooth wait time measurement system installed at the Peace Bridge and the Lewiston-Queenston Bridge crossings uses Traffax readers and FastLane BluFaxWeb software for computing the average wait time by vehicle time and direction (Roelofs et al., 2016).

Bluetooth is a telecommunications industry specification that defines the way in which the digital devices can interconnect easily using short-range wireless communications ("tpa-na Traffic Monitoring," n.d.). The BluFax monitoring system can collect the travel times by sampling a portion of actual travel times from the traffic stream ("tpa-na Traffic Monitoring," n.d.). It measures the travel times by matching the MAC addresses of Bluetooth devices at two different locations ("tpa-na Traffic Monitoring," n.d.).

As stated at tpa-na.com ("tpa-na Traffic Monitoring," n.d.), Bluetooth devices have the following advantages over the existing methods:

- Unlike the existing point detection technology, which includes inductive loops, radar detectors, image processors, etc., Bluetooth technology measures the travel time directly by the equipment due to which it has greater accuracy ("tpa-na Traffic Monitoring," n.d.).
- It can be applied globally because of the proliferation of the Bluetooth standard protocol ("tpa-na Traffic Monitoring," n.d.).

- It can measure the travel times for different modes like highway vehicles, rail, and pedestrians because the Bluetooth devices are associated with people, not the vehicle ("tpa-na Traffic Monitoring," n.d.).
- As there are no databases of Bluetooth addresses, Bluetooth technology offers more privacy than the toll tag tracking, cellular telephone geolocation, or license plate surveys ("tpa-na Traffic Monitoring," n.d.).
- The field installation procedure is simple ("tpa-na Traffic Monitoring," n.d.).

The red pointers in figure 3.1 – 3.3, shows the location of Bluetooth reader installation at the different bridges. The information about this location was provided by NITTEC.

Figure 3. 1: Location of Bluetooth readers at Peace Bridge

Figure 3. 2: Location of Bluetooth readers at Rainbow Bridge

Figure 3. 3: Location of Bluetooth readers at Queenston Lewiston Bridge

3.3 Deep Learning

A Deep learning technique is a type of Machine learning and an Artificial intelligence (AI) approach (Goodfellow, Bengio, & Courville, 2016). Skansi (Skansi, 2018) has stated that the machine learning having deep artificial neural networks is called deep learning. Deep learning is a method for training models through various levels of abstraction (Alpaydin, 2016). A model is a template that gives the relationship between the inputs and outputs, even though its structure is fixed, it does have some modifiable parameters that can be adjusted to give a different relation when trained on a different data (Alpaydin, 2016). Figure 3.4 shows the relationship between AI, machine learning and deep learning.

To understand deep learning, it might be necessary to understand some concepts like neural networks and machine learning. A neuron is a simple processing unit and the network of these neurons and the connections between them are called a neural network (Alpaydin, 2016). Machine learning refers to the capability of the systems to gain their own knowledge by extracting the patterns from the raw data (Goodfellow et al., 2016). Supervised and unsupervised learning algorithms are two broad categories of machine learning algorithms (Goodfellow et al., 2016). The learning algorithms that experiences a dataset with examples having a label or target are called supervised learning algorithms, while those which have to learn the properties of the structure from the dataset provided to them are called unsupervised learning algorithms (Goodfellow et al., 2016). Generally, a machine learning algorithm involves some hyperparameters like the number of hidden units, learning rate, dropout rate, convolutional kernel width, implicit zero padding, etc. (Goodfellow et al., 2016). The learning algorithm cannot adjust the hyperparameters by itself, however, their settings can control the behavior of the algorithm (Goodfellow et al., 2016).

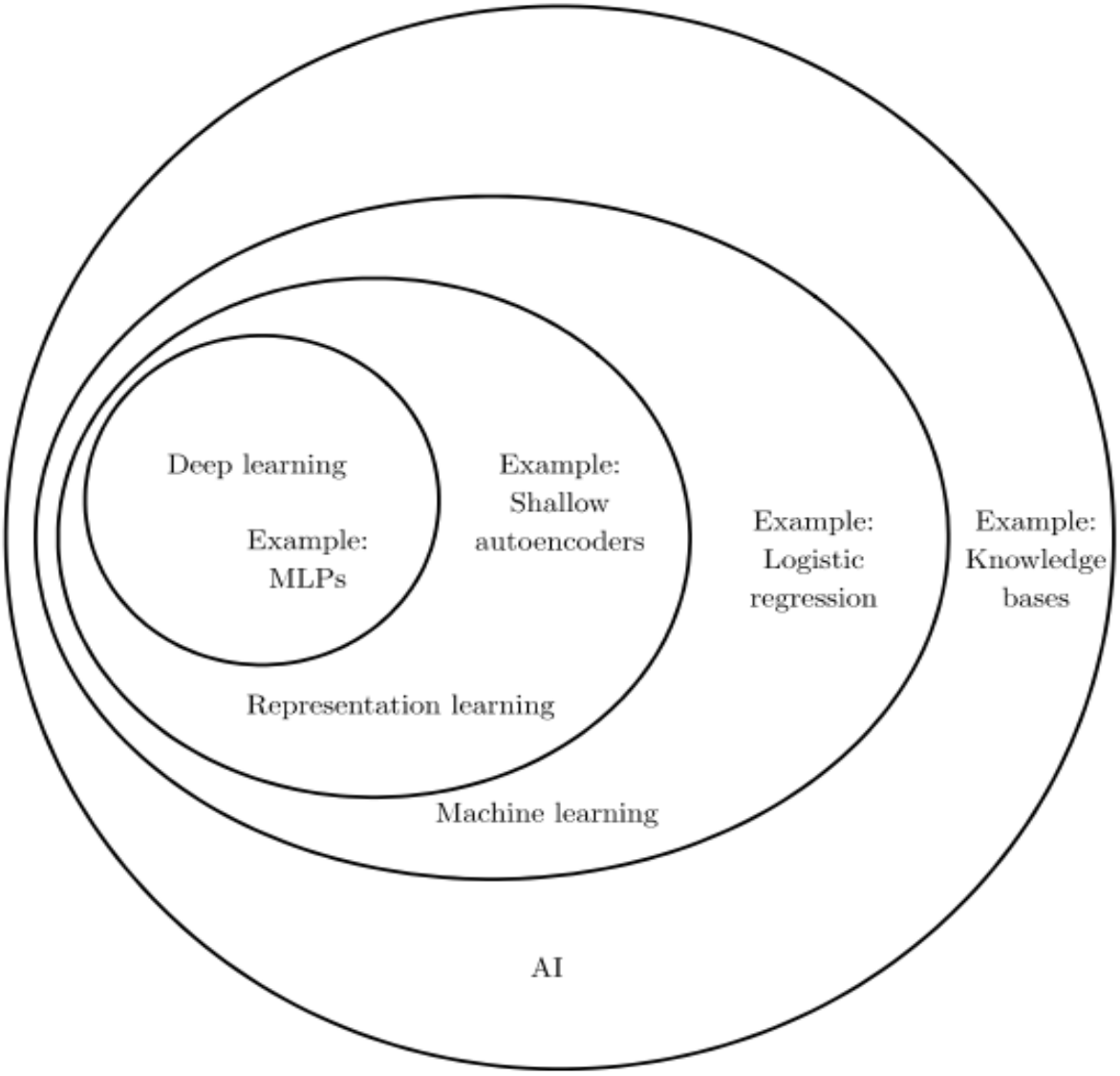


Figure 3. 4: Venn diagram showing the relationship between AI, machine learning, and deep learning (Goodfellow et al., 2016)

Goodfellow et. al. (Goodfellow et al., 2016) have mentioned that deep learning has a long history which can be traced back to the 1940s. They have described that deep learning has mainly experienced three waves of development and was known by different names each time, these include the first wave from the 1940s to 1960s, the second wave from 1980s to 1990s and the third wave from 2006 (Goodfellow et al., 2016). Deep learning was known as cybernetics and

connectionism during the first and second wave respectively (Goodfellow et al., 2016). Neuroscience is considered to be an inspiration for deep learning researchers, however, the modern deep learning doesn't hold neuroscience as its predominant guide (Goodfellow et al., 2016).

The usefulness of deep learning has increased with a greater amount of training data available and it has been able to solve progressively more complicated applications more accurately with time (Goodfellow et al., 2016). As per Khan et. al. (S. Khan, Rahmani, Shah, & Bennamoun, 2018), the advantages of deep learning include the simplicity in generating large networks of deep learning and their easy scalability to huge datasets. Deep learning has been applied to various fields like robotics, natural language processing, search engines, online advertising, video games, and finance (Goodfellow et al., 2016).

3.4 Deep Learning Techniques

Each deep learning technique works differently. Various processes take place during the prediction of the next values by these techniques. In this section, the theory and working of each deep learning technique are explained in detail.

3.4.1 Multilayer Perceptron (MLP)

Multilayer Perceptron (MLP) is one of the deep learning techniques. It comprises of three components, namely input layer, hidden layer, and an output layer (Pal & Prakash, 2017). Each layer contains several numbers of neurons or nodes (Gardner & Dorling, 1998). As explained by Gardener and Dorling (Gardner & Dorling, 1998), MLPs consists of a system of neurons interconnected by weights (w). They have stated that MLPs are fully connected as each neuron is connected to every other neuron in the previous and next layer. They have further explained that

MLPs could have one or more hidden layers and their architecture is not fixed. Figure 3.5 shows an illustration of MLP.

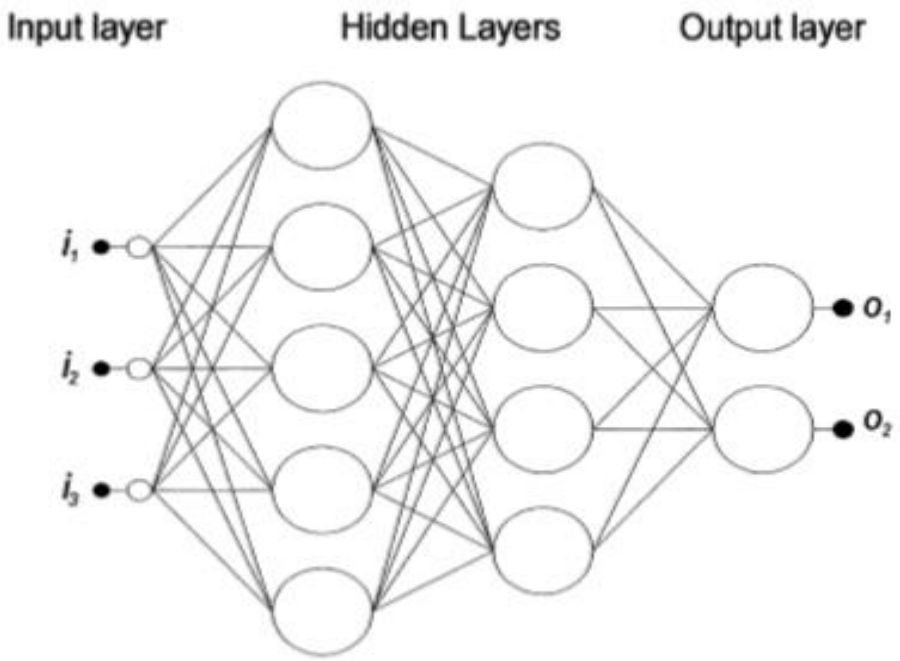


Figure 3. 5: MLP with two hidden layers (Gardner & Dorling, 1998)

Training of MLPs is a process of determining the individual weights such that the relationship that has to be modeled is accurately resolved (Gardner & Dorling, 1998). Gradient Descent is a technique that is used by the backpropagation training algorithm to train the MLPs (Gardner & Dorling, 1998).

Pal and Prakash (Pal & Prakash, 2017) have described in detail the training process of MLP models. They have explained that the input features are fed from the input layer into the hidden layers, where each neuron applies a linear transformation and a non-linear activation to the input features. They demonstrated that the output (g_i) from each of these neurons is:

$$g_i = h\ (w_i x + b_i)$$

where wi and bi are the weights and bias of the linear transformation respectively and h is an activation function. They have pointed out that MLPs can model the non-linear relationship between the regressors and target variable with the help of non-linear activation function (Pal & Prakash, 2017). As per Skansi (Skansi, 2018), the most common activation function is sigmoid or logistic function, which outputs $\sigma(z)$ equal to $1/(1+e^{-z})$, where z (also called logit) is the sum of the product of inputs to the neuron with their respective weights plus bias. Bias is the modifiable value in each neuron (Skansi, 2018). As explained by Pal and Prakash (Pal & Prakash, 2017), the output from the neurons of one hidden layer are fed as an input into the next hidden layer, where again transformations of the inputs take place and the outputs are fed into the next layer and this procedure goes on till the last hidden layer feeds the output layer. The process of transformation of the input layer to prediction is known as the forward pass (Pal & Prakash, 2017). They have further explained that after the forward pass is completed, loss or error (E) is computed, which is the difference between the predicted value and the target value. Mean squared error (MSE) or mean absolute error (MAE) is most suitable for training the models for time series prediction (Pal & Prakash, 2017). Next, the backpropagation algorithm is applied to compute the partial derivatives of loss with respect to the weights ($\partial E/\partial w$) in the backward direction, i.e. beginning from the output layer and going up to the input layer, this is known as backward pass (Pal & Prakash, 2017). Finally, the weights of connections between each neuron, which were randomly initiated are now updated based on the learning rate and the results obtained from the backward pass (Pal & Prakash, 2017). As stated by Skansi (Skansi, 2018), the weights updated by the equation:

$$w_i^{new} = w_i^{old} + (-1)\, \eta\, \partial E/\partial\, w_i^{old}$$

where w_i^{new} is the new updated weight, w_i^{old} is the old weight and η is the learning rate

As explained by Gardener and Dorling (Gardner & Dorling, 1998), thousands of training iterations might be required for obtaining a MLP model with an acceptable level of error but the training should be stopped when the performance of the model reaches maximum on the independent test set. The weights are updated after each iteration (Pal & Prakash, 2017). The number of times the iterative weight update is repeated is called epochs (Pal & Prakash, 2017). Through the iterative process of the forward and backward pass, MLPs could understand the relationship between the dependent and the target variable and can also make predictions.

As the direction of information processing in MLP is from the input layer to the output layer, they are called the feed-forward neural network (Gardner & Dorling, 1998).

3.4.2 Convolutional Neural Network (CNN)

Convolutional Neural Network or CNN is another deep learning method. Aghdam and Heravi (Aghdam & Heravi, 2017) have pointed out that applying a fully connected feedforward network on an image will result in a huge number of neurons, which makes them impractical for usage and therefore, the basic idea behind CNNs is to build a deep network with few numbers of parameters. The two types of convolutional layers in CNNs are 1 D convolutional layers or temporal convolutional layer, and 2 D convolutional layers or planar convolutional layers (Skansi, 2018). 2 D convolutions are generally applied to images, while 1 D convolutions are usually applied on sequential inputs (Pal & Prakash, 2017).

Aghdam and Heravi (Aghdam & Heravi, 2017) have stated that a CNN generally comprises of several convolution-pooling layers that are followed by fully connected layers. Figure 3.6 shows a diagrammatic representation of CNN. The convolutional layers usually contain multiple filters, which moves over the entire image, this movement is called convolution (Pal & Prakash, 2017).

As per Khan et.al. (S. Khan et al., 2018), each filter is a grid of discrete numbers, which are also called the weight of the filter. They have further stated that the number of steps of the filter along the horizontal or vertical direction is called stride of the convolutional filter. They have demonstrated the convolutional operation that results in output feature maps due to the convolution between the filters and the inputs to the convolution layer. Figure 3.7 shows the convolution operation of a 2 X 2 filter with a 4 X 4 input feature map to produce a 3 X 3 output feature map (S. Khan et al., 2018). Pal and Prakash (Pal & Prakash, 2017) have explained that the summation of the product of weights of the filter and the corresponding pixel values of the image plus a bias (optional) is the final feature from a local patch which results into a value of output feature map. Aghdam and Heravi (Aghdam & Heravi, 2017) have explained the weight sharing property of CNNs due to which the neurons in the same filter share the same set of weight, which results into a decrease in the number of parameters.

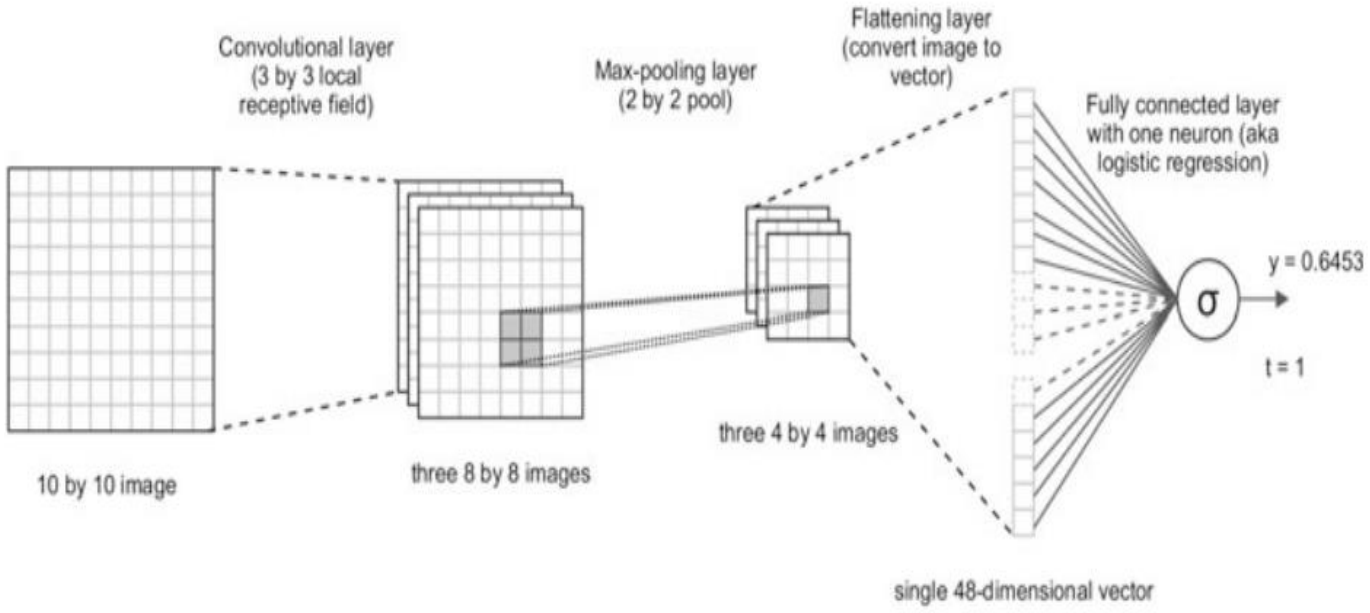


Figure 3. 6: A typical CNN applied to a 2 D image (Skansi, 2018)

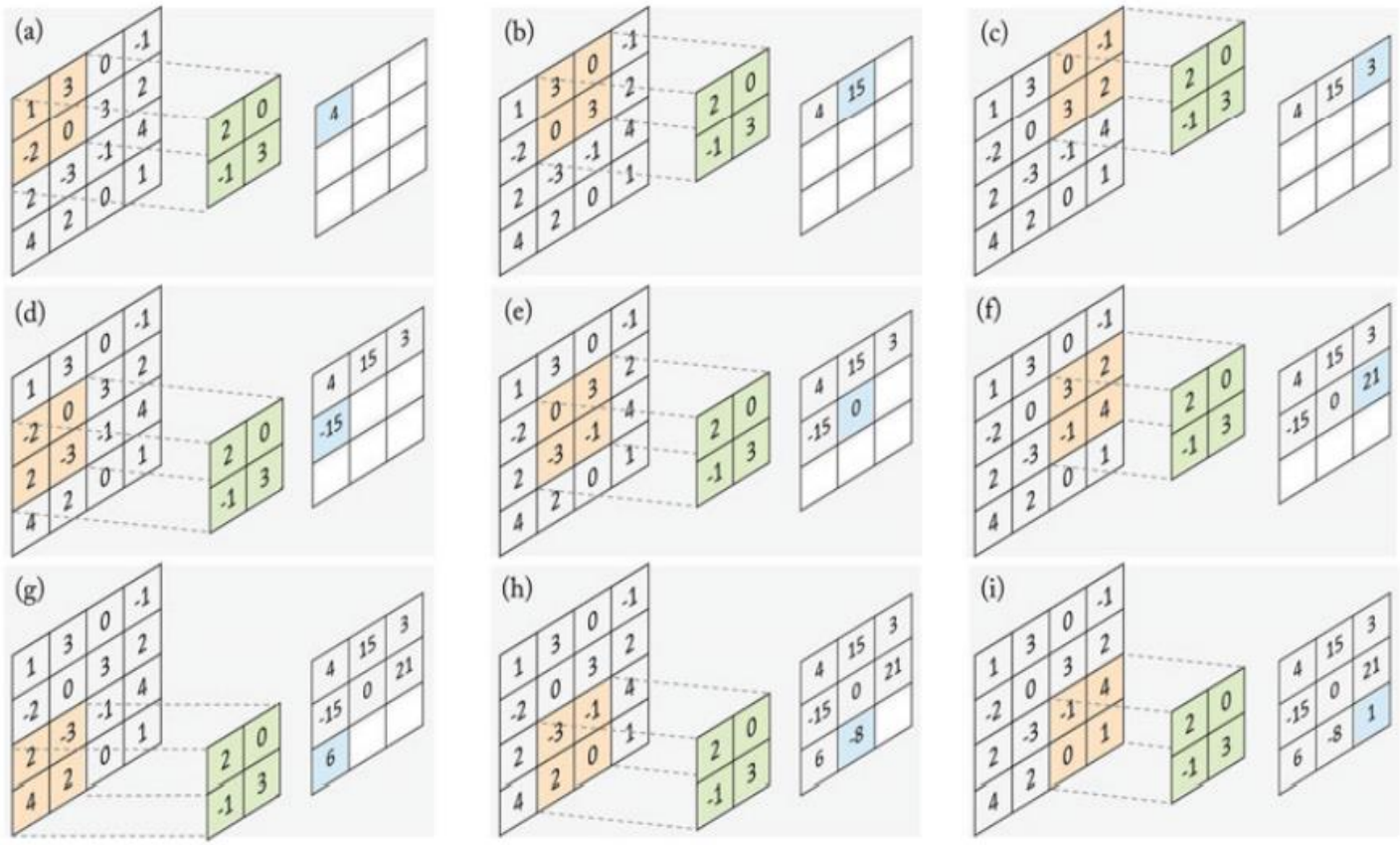


Figure 3. 7: Stepwise operation of the convolutional layer with 2 X 2 filter and stride = 1 (S. Khan et al., 2018)

Khan et. al. (S. Khan et al., 2018) have pointed out that the spatial size of the output feature map obtained after convolution may be smaller than the input feature map, to avoid zero padding may be applied, which means increasing the size of input feature map each direction by padding zeroes so as to output feature map with desired size.

As per Khan et. al. (S. Khan et al., 2018), the dimensions of the output feature map (h' X w') from convolution operation is given by

$$h' = \{h - f - (d - 1)(f - 1) + s + 2p\}/s$$

$$w' = \{w - f - (d - 1)(f - 1) + s + 2p\}/s$$

where, h X w is the size of the input feature map, filter in the convolutional layer has size f x f, d is dilation factor, s is stride and p is the increase in input feature map in each dimension due to zero padding.

The convolutional layers and fully connected layers in a CNN are generally followed by a non-linear activation function which enables the network to learn nonlinear mappings (S. Khan et al., 2018). As stated by Pal and Prakash (Pal & Prakash, 2017), rectified linear units (ReLu) is the popular choice for activation function, which is given by:

$$\text{ReLu}(z) = 0, \text{ if } z \leq 0$$

$$= z, \text{ if } z > 0$$

Before the output from the convolutional layers is fed to dense layers, they may be passed through pooling layer (Pal & Prakash, 2017). The purpose of pooling layer is downsampling, which means to decrease the dimensionality of the feature map (Aghdam & Heravi, 2017). The pooling layers conduct combination operation on the blocks of the input feature map, which is defined by a pooling function like max or average pooling (S. Khan et al., 2018). A window of prespecified size and stride is moved across input feature map and pooling operation like max pooling takes place in which the maximum value from the selected block of input feature map is chosen (S. Khan et al., 2018). There are no trainable weights in the pooling layer (Pal & Prakash, 2017). As stated by Khan et. al. (S. Khan et al., 2018), the size (h' X w') of the output feature map from the pooling layer is given by:

$$h' = |\_(h - f + s)/s\_|$$

$$w' = |\_(w - f + s)/s\_|$$

where, the size of the input feature map is h X w, size of pooling region is f X f, the stride is s, and $\lfloor \cdot \rfloor$ represents floor operation.

As per Khan et. al. (S. Khan et al., 2018), Fully connected layers are generally added near the end of the architecture in a typical CNN and their operation can be represented by:

$$y = f(W^T x + b)$$

where y is the vector of output activations and x is the vector of input activation, W is the matrix of weights of the connections between the layer units, b is bias and f (·) is a nonlinear function.

Khan et. al (S. Khan et al., 2018) have explained that during the training process of CNN, the parameters of CNN are optimized such that the loss function is minimized. These parameters are the tunable weights in the layers of CNN (S. Khan et al., 2018). They have further explained that gradient based methods are used for the iterative search of the locally optimal solution at each step and to update the parameters in the direction of steepest descent (S. Khan et al., 2018). As the information is pushed forward, CNNs are also called feedforward neural networks (Skansi, 2018).

Skansi (Skansi, 2018) has suggested that the CNNs are easier to train because they require less number of parameters. He also pointed out that due to the shared same of weights in CNN, the problem of vanishing gradient is avoided as the same weights get updated each time even if just a little bit. Additionally, the process involved in CNNs is computationally fast and can be split across many processors, which is due to the fact that training of each feature map can be done parallelly (Skansi, 2018).

3.4.3 Recurrent Neural Network (RNN)

Networks that have feedback loops in which some connections feed the output back into a layer as input are known as Recurrent Neural Networks or RNNs (Skansi, 2018).

The diagrammatic representation of a simple RNN architecture is shown in figure 3.8, where the circles with x, y, and h denotes input, output, and hidden nodes, while the squares with W^{h}_{i}, W^{o}_{h}, and W^{h}_{h} are matrices representing the input, output, and hidden weights respectively, and the polygon denotes nonlinear transformation (Bianchi et al., 2017).

The procedure of representing RNN as an infinite, acyclic and directed graph is known as unfolding (Bianchi et al., 2017). It comprises of replicating the hidden layer structure of the network for each time step (Bianchi et al., 2017). Figure 3.9 shows the unfolded RNN. As explained by Bianchi et. al. (Bianchi et al., 2017) that unlike the standard feedforward neural networks, the weight matrices of unfolded RNNs are constrained to assume same values in all replicas of the layer. They have stated that due to this transformation, a direct relation between network weights and the loss function can be found and hence, the network can be trained with a standard learning algorithm.

As explained by Bianchi et. al. (Bianchi et al., 2017), the training procedure of RNN may be based on backpropagation through time (BTT) for propagation and distribution of prediction error to the previous states of the network. BTT is a special case of the backpropagation algorithm (Pal & Prakash, 2017). Usually, the training of neural networks involves using a gradient descent algorithm for updating its parameters so as to minimize the loss function (Bianchi et al., 2017).

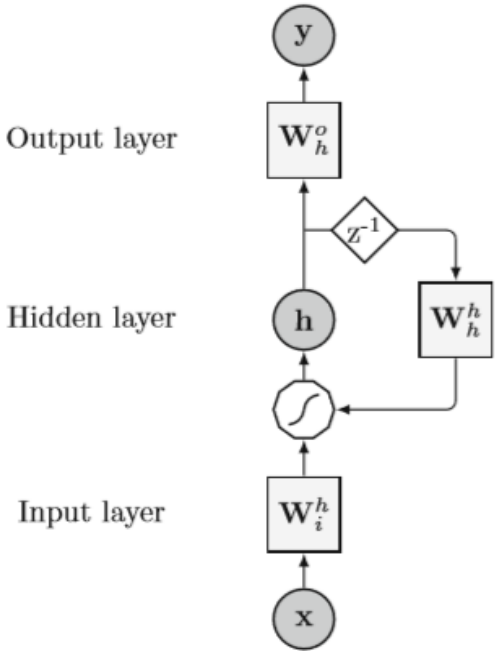


Figure 3. 8: A simple RNN architecture (Bianchi et al., 2017)

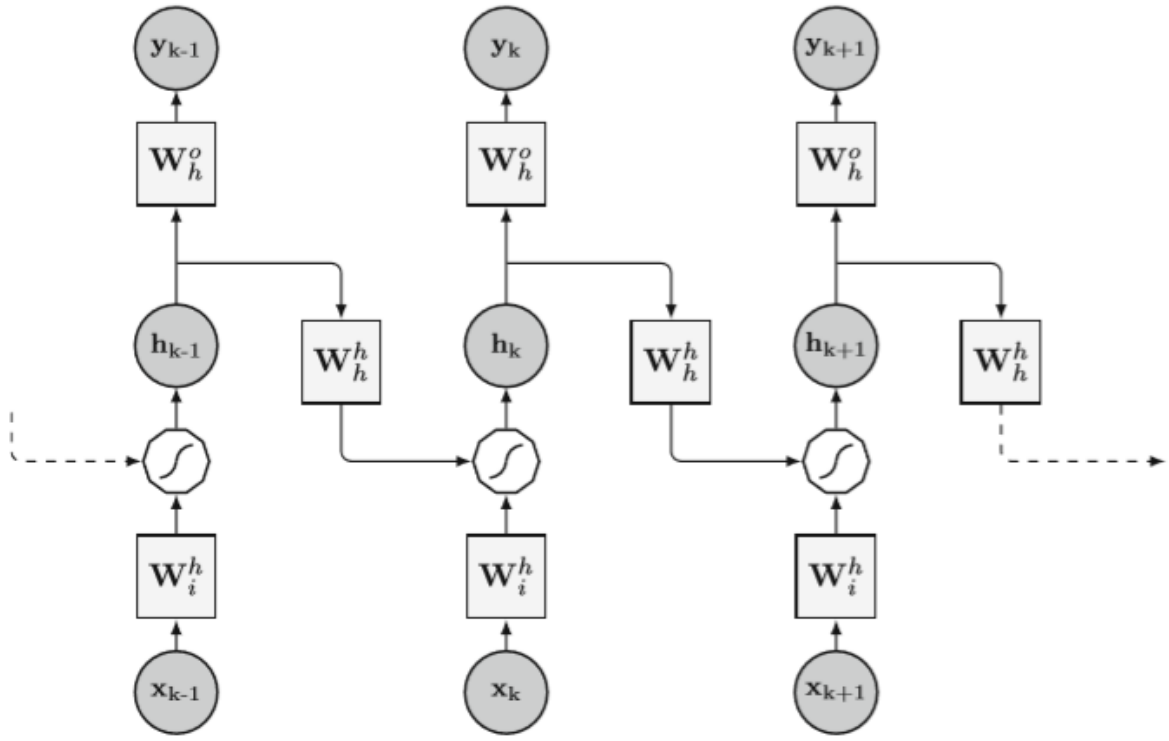


Figure 3. 9: RNN unfolded to feedforward neural network (Bianchi et al., 2017)

Pal and Prakash (Pal & Prakash, 2017) have described the process to calculate BTT. The loss (L) or error between the predicted and target variable is found through forward pass and then the partial derivative of loss with respect to the network weights ($\partial L/\partial W$) is computed while going in the backward direction, i.e. from the loss to the weight (Pal & Prakash, 2017). However, there can be several paths connecting the loss to the weight as RNN has a sequential structure (Pal & Prakash, 2017). Therefore, the partial derivative ($\partial L/\partial W$) is computed as the summation of partial derivatives along each path from the loss node to every time step node and this technique is called BTT (Pal & Prakash, 2017). Although, the multiplicative terms used in the computation of gradient may be fractional and in long-range timesteps, the product of these terms might reduce the gradient to zero or negligibly low, which does not allow the weights to update (Pal & Prakash, 2017). This problem is called vanishing gradient. (Pal & Prakash, 2017).

The different types of recurrent neural network architecture - Elman Recurrent Neural Network (ELNN), Long Short-term Memory (LSTM), and Gated Recurrent Unit (GRU), are described below in detail.

3.4.3.1 Elman Recurrent Neural Network (ELNN)

Elman Recurrent Neural Network is believed to be the most basic version of RNN and is also called Simple RNN or Vanilla RNN (Bianchi et al., 2017).

Figure 3.8 shows the architecture of an ELNN. It comprises of input and output layers with feedforward connections, and hidden layers with recurrent connections (Bianchi et al., 2017). ELNN also have h which are the input for the recurrent connection (Skansi, 2018). As stated by Bianchi et. al. (Bianchi et al., 2017), at time t, the update of the internal state and the output of the network is given by:

$$h[t] = f\,(W^{h}_{i}\,(x[t] + b_i) + W^{h}_{h}\,(h[t - 1] + b_h)$$

$$y[t] = g\,(W^{o}_{h}\,(h[t] + b_o)$$

where, W^{h}_{i}, W^{o}_{h}, and W^{h}_{h} are matrices representing the input, output, and hidden weights, respectively; x[t] is the input, y[t] is the output of the network; h[t] is the internal state, b_i, b_h, and b_o, are bias vectors and f (·) is the activation function of the neuron. (Bianchi et al., 2017). h[t] is generally initialized as a vector of zeroes, and it transfers the memory contents of the network at time t (Bianchi et al., 2017).

Pal and Prakash (Pal & Prakash, 2017) have stated that ELNNs suffer due to vanishing and exploding gradients.

3.3.3.2 Long Short-Term Memory Recurrent Neural Networks (LSTM RNN)

As ELNN faces difficulty in effectively learning the long-range dependencies because of vanishing and exploding gradients, LSTMs were developed to resolve this issue (Pal & Prakash, 2017). LSTMs can accurately model long-term and short-dependencies in the data (Bianchi et al., 2017). They do not impose any bias towards recent observations and allow the constant error to flow back through time, and by doing this, it tries to resolve the issue of vanishing gradients (Bianchi et al., 2017).

Bianchi et. al. (Bianchi et al., 2017) have explained that unlike the ERNNs, LSTMs apply a more elaborate internal processing unit, which is called the cell. They have further explained that a LSTM cell is made up of five different nonlinear components interacting in a definite manner. Additionally, a cell's internal state is modified only by linear interactions, which allows smooth backpropagation of information across time (Bianchi et al., 2017). Figure 3.10 shows a cell in

LSTM, where, x_t and y_t are the external input and external output of the cell respectively; h_{t-1}, h_t, y_{t-1}, and y_t are internal state variables; g_1 and g_2 are operators with nonlinear transformation, and σ_f, σ_u, and σ_o are sigmoid in forget, update, and output gate respectively (Bianchi et al., 2017).

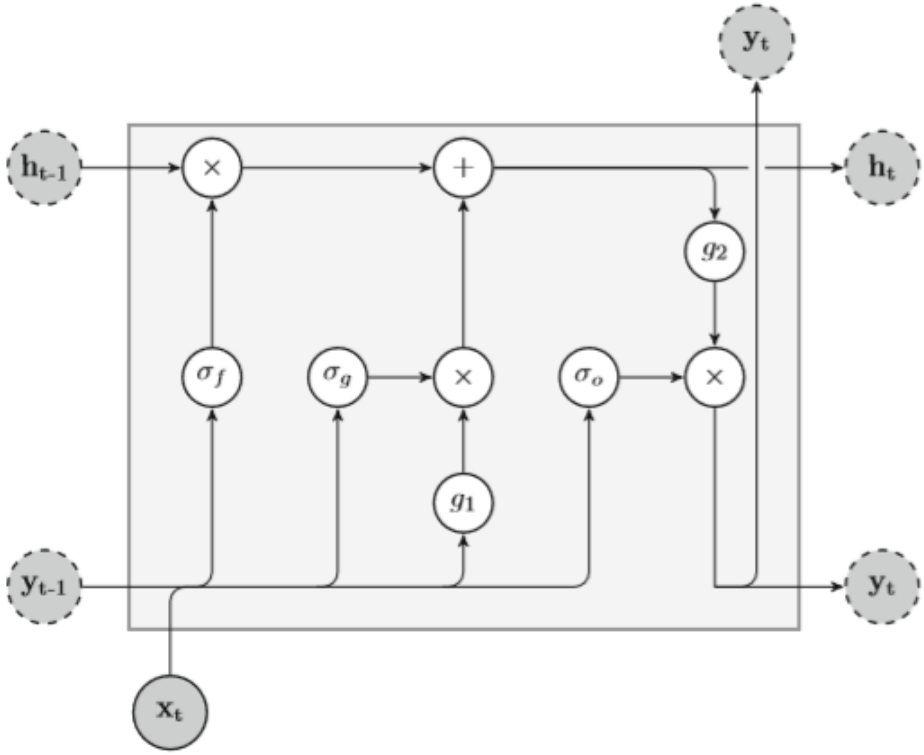


Figure 3. 10: Cell of a LSTM (Bianchi et al., 2017)

For protecting and controlling information in the cells, LSTM uses three gates, which are forget gate, input gate and output gate (Bianchi et al., 2017). The information that must be removed from the previous cell state h[t-1] is decided by the forget gate, how much the new state h[t] must be updated by new candidate ĥ[t] is decided by input gate, and the part of the state to be outputted is decided by the output gate (Bianchi et al., 2017).

For updating the cell state and computing the output, the difference equations of forward pass, as given by Bianchi et. al. (Bianchi et al., 2017) are:

$$\text{forget gate: } \sigma_f[t] = \sigma\,(W_f\, x[t] + R_f y[t-1] + b_f)$$

$$\text{candidate state: } \tilde{h}[t] = g_1(W_h x[t] + R_h y[t-1] + b_h)$$

$$\text{input gate: } \sigma_u[t] = \sigma\,(W_u x[t] + R_u y[t-1] + b_u)$$

$$\text{cell state: } h[t] = \sigma_u[t]\,(\cdot)\,\tilde{h}[t] + \sigma_f[t]\,(\cdot)\,h[t-1]$$

$$\text{output gate: } \sigma_o[t] = \sigma(W_o x[t] + R_o y[t-1] + b_o),$$

$$\text{output: } y[t] = \sigma_o[t]\,(\cdot)\,g_2(h[t])$$

where, x[t] is the input vector to the cell at time t; $\sigma(\cdot)$ denotes a sigmoid function; $g_1(\cdot)$ and $g_2(\cdot)$ are point wise nonlinear activation function, and $(\cdot)$ is the entry wise multiplication between two vectors; W_o, W_u, W_h, and W_f are weight matrices applied to input of the cell; R_o, R_u, R_h, and R_f are weights matrices of the recurrent connections; b_o, b_u, b_f, and b_h are bias vectors (Bianchi et al., 2017).

However, as per Skansi (Skansi, 2018), the interpretations of LSTM are just metamorphic and it very rarely works like a human brain. Bianchi et. al (Bianchi et al., 2017), have also pointed out that practically, the forget and update gate of LSTM never opens or closes totally and contents of the cell may change over time.

3.4.3.3 Gated Recurrent Unit Recurrent Neural Network (GRU RNN)

Introduced in 2014, GRUs are a simpler version of LSTM and can resolve the issue of long-term dependencies in ELNNs (Pal & Prakash, 2017). GRUs are capable of adaptably capturing the dependencies at different time scales (Bianchi et al., 2017). As stated by Pal and Prakash (Pal &

Prakash, 2017), GRU has fewer trainable weights than LSTM. Figure 3.11 shows the architecture of a GRU cell.

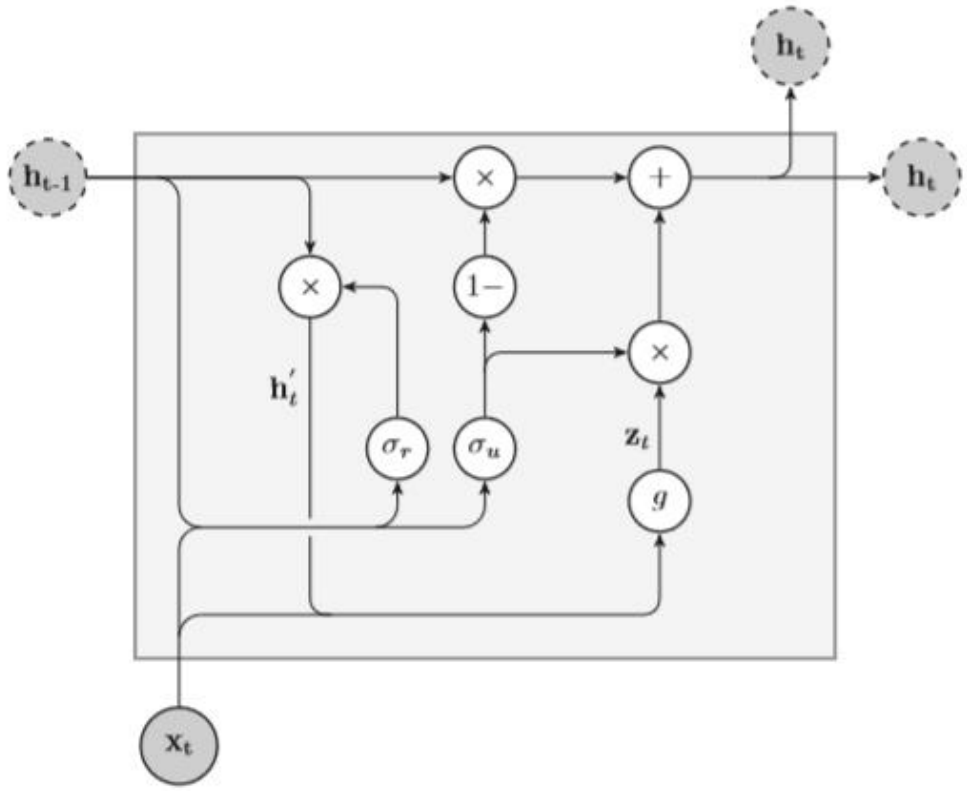


Figure 3. 11: Cell of a GRU (Bianchi et al., 2017)

GRU has two gates, which are update gate, and reset gate (Pal & Prakash, 2017). The forget and input gates in LSTM are combined to form a single update gate in GRU (Bianchi et al., 2017). The function of update gate is to adaptively decide how much must be remembered or forgotten by the hidden units, while the memory of a cell can be reset by the reset gate (Bianchi et al., 2017).

The state equations of GRU given by Bianchi et. al. (Bianchi et al., 2017) are:

$$\text{reset gate: } r[t] = \sigma(W_r h[t-1] + R_r x[t] + b_r)$$

$$\text{current state: } h'[t] = h[t-1] (\cdot) r[t]$$

$$\text{candidate state: } z[t] = g(W_z h'[t-1] + R_z x[t] + b_z)$$

$$\text{update gate: } u[t] = \sigma(W_u h[t-1] + R_u x[t] + b_u)$$

$$\text{new state: } h[t] = (1-u[t]) (\cdot) h[t-1] + u[t] (\cdot) z[t]$$

where, $\sigma(\cdot)$ denotes a sigmoid function; $g_1(\cdot)$ and $g_2(\cdot)$ are point wise nonlinear activation function, and $(\cdot)$ is the entry wise multiplication between two vectors; W_u, W_z, W_r, R_u, R_z, and R_r are weights matrices of the recurrent connections; b_u, b_z, and b_r are bias vectors (Bianchi et al., 2017).

3.5 Optimization

Some kind of optimization is generally involved in the deep learning algorithms, however, the optimization algorithm used in deep models is quite different from the traditional optimization algorithms (Goodfellow et al., 2016). Unlike in pure optimization, in machine learning, a cost function $J(\theta)$ is reduced in hope to improve some performance measure P, which is defined based on the test set (Goodfellow et al., 2016). An optimization algorithm is known as batch or deterministic gradient method if it uses the whole training set, while it is called stochastic or online method if only one example is used at a time (Goodfellow et al., 2016). Additionally, the optimization algorithm is called minibatch or minibatch stochastic or simply stochastic method if the training examples used are more than one but not all (Goodfellow et al., 2016). The optimization algorithms that were tried in the deep learning models developed for this study are briefly discussed below.

I. Stochastic Gradient Descent (SGD): In contrast to gradient descent which follows to downhill the gradient of the whole training set, SGD just uses randomly selected minibatches instead of the whole training set and hence, significantly accelerates the

process (Goodfellow et al., 2016). Learning rate is the step size of following downhill and may be selected by trial and error or by observing the learning curve which plots the objective function as a function of time (Goodfellow et al., 2016). As per Goodfellow et. al. (Goodfellow et al., 2016), in SGD, the initial parameter θ is updated in iteration k as $\theta - \epsilon\hat{g}$, where ϵ is the learning rate in iteration k, $\hat{g}$ is gradient estimate which is equal to 1/m times the gradient of loss function L with respect to θ, which can be mathematically represented as:

$$\hat{g} \leftarrow +(1/m)\nabla_\theta \sum_i L(f(x^{(i)};\theta),y^{(i)})$$

$$\theta \leftarrow \theta - \epsilon\hat{g}$$

where $y^{(i)}$ is the target corresponding to a minibatch $\{x^{(1)}, ..., x^{(m)}\}$ of m examples taken from the training set (Goodfellow et al., 2016).

II. Optimization algorithms with adaptive learning rates: These include -

a. AdaGrad: As per Khan et. al. (S. Khan et al., 2018), AdaGrad or Adaptive gradient is an algorithm which uses adaptive learning rate for each parameter i, such that the parameter θ_i at time step t is:

$$\theta^i_t = \theta^i_{t-1} - \eta\delta^i_t / \sqrt{(\sum^t_{\tau=1} (\delta^i_t)^2 + \epsilon)}$$

where η is the learning rate which is fixed during the training phase, δ^i_t is the gradient with respect to the parameter θ_i at t time-step, $\sum^t_{\tau=1} (\delta^i_t)^2$ is the summation of the squares of all historic gradients for parameter θ_i and ϵ is an extremely small term (S. Khan et al., 2018).

b. AdaDelta: Khan et. al. (S. Khan et al., 2018) have stated that in case of AdaGrad, an initial learning rate has to be set and it suffers from vanishing learning rate problem in which the

effective learning rate gets very small when t increases, these issues are solved by AdaDelta or Adaptive delta algorithm. They have further explained that AdaDelta uses only the last few gradients for computing θ^i_t instead of all historic gradients as in AdaGrad. As per Khan et. al. (S. Khan et al., 2018), in AdaDelta, θ^i_t is updated as:

$$\theta^i_t = \theta^i_{t-1} - [\delta^i_t * \sqrt{(E[(\Delta\theta)^2]_{t-1} + \epsilon)} / \sqrt{(E[\delta]^2_t + \epsilon)}$$

where $\Delta\theta = \theta^i_t - \theta^i_{t-1}$, $E[\delta]^2_t$ is the running average and is equal to $\gamma E[\delta^2]_{t-1} + (1-\gamma)\ \delta_t{}^2$, and γ is called decay (S. Khan et al., 2018).

c. RMSProp: As explained by Khan et. al. (S. Khan et al., 2018), RMSProp is another optimization algorithm which tries to solve the vanishing learning rate problem of AdaGrad and the update rule of tunable parameters is given by:

$$\theta^i_t = \theta^i_{t-1} - \eta\delta^i_t / \sqrt{(E[\delta]^2_t + \epsilon)}$$

d. Adam: Adam or Adaptive Moment Estimation is an optimization algorithm that tries to resolve the vanishing learning rate problem and is suitable even when the gradients are sparse (S. Khan et al., 2018). As per Khan et. al. (S. Khan et al., 2018), its update rule is:

$$\theta^i_t = \theta^i_{t-1} - \eta\hat{E}[\delta]_t / [\sqrt{(\hat{E}[\delta]^2_t)} + \epsilon]$$

where $\hat{E}[\delta]_t = E[\delta]_t / (1-\gamma_1)^t$ and $\hat{E}[\delta]^2_t = E[\delta^2]_t / (1-\gamma_2)^t$

3.6 Regularization

For machine learning models, the error measured on the training set is called training error, while that measured on the unobserved or new inputs is called generalization error or test error (Goodfellow et al., 2016). As per Goodfellow et. al. (Goodfellow et al., 2016), underfitting and overfitting are two challenges in machine learning. Underfitting is said to occur when the model is unable to get sufficiently low training error while overfitting occurs when there is a big gap between the training error and test error (Goodfellow et al., 2016).

As described by Goodfellow et. al. (Goodfellow et al., 2016), regularization refers to any change made in the learning algorithm to decrease its generalization error, but not the training error. There are various techniques described in the literature for applying regularization. Some of the regularization strategies for deep learning models discussed by Goodfellow et. al. (Goodfellow et al., 2016) includes parameter norm penalties, norm penalties as constrained optimization, data augmentation, noise robustness, early stopping, parameter tying and parameter sharing, dropout, and adversarial training. Dropout technique was used in the deep learning models that were developed in this study for predicting short-term delay at border crossings. Hence, in this section, only the regularization by dropout is discussed.

Dropout is the technique used for improving the learning of neural networks and reduce overfitting (Skansi, 2018). Dropout can be applied by adding the dropout parameter π having a value between 0 and 1, and by doing so every weight will be set to zero with a probability of π in each epoch (Skansi, 2018). As per Goodfellow et. al. (Goodfellow et al., 2016), the advantages of dropout are that it is very computationally inexpensive and it does not restrict much the choice of model or training procedure that can be used.

3.7 Model Development

In this study, the short-term car traffic delay was predicted by forecasting time series using the deep learning methods, namely Multilayer Perceptron (MLP), Convolutional Neural Networks (CNN), Long Short-Term Memory Recurrent Neural Networks (LSTM RNN), and Gated Recurrent Unit Recurrent Neural Networks (GRU RNN).

Each modeling dataset (complete set, weekday set, and weekend set) was split into train set, validation set, and test set. The train set was used for training the models. The validation set was

used for selecting the hyperparameters by comparing the performance of various models to select the best one. Lastly, the test set was used to report the results from the selected model. The first 60% of the dataset formed the train set, the next 20% was used as a validation set and the last 20% as a test set.

The dataset that was fed into the model was a time series of traffic delay (in minutes) at every 5 minutes interval. The model takes traffic delay at prior time steps as its input and outputs the traffic delay predicted for the future. The model was developed such that it outputs the predicted delay for the next 12 time steps. Therefore, the model output is the predicted traffic delay for the next 5 minutes, 10 minutes, 15 minutes, and so on till the next 60 minutes in the future. The number of prior time steps that the model takes as input can be varied. Mathematically, if the time is t and the model is set to take n number of prior time steps then the model inputs will be the delay at time t, t-5, t-10, and so on up to n time steps. Whereas, the model outputs will be the delay at time t+5, t+10, t+15, and so on till t+60.

The models developed trained using the whole data or complete data are called 'complete set' models. The models trained only on weekday data are called 'weekdays' model and those trained only on weekend data are called 'weekends' model. The predicted delays obtained from the models were compared with the actual delay to evaluate the predictive accuracy of deep learning models, compare the performance of the four deep learning techniques and to find the effect of data classification on model's prediction accuracy.

The models developed for this study were build using Keras (v2.2.2) deep learning library, backend by TensorFlow (v1.9.0) in the programming language Python (v3.5.6) on a computer with 8.00 GB RAM and Intel®Core™i56200U CPU. The various libraries used in the python codes

include scikit-learn, numpy, matplotlib, pandas, and math. The python codes used in this study for developing and comparing the deep learning models are inspired by the codes presented by (Brownlee, 2016, 2018; Pal & Prakash, 2017). Additionally, (Ardit, 2017; "Customizing Ticks | Python Data Science Handbook," n.d.; "Home - Keras Documentation," n.d.; "Matplotlib: Python plotting — Matplotlib 3.0.3 documentation," n.d.; "scikit-learn: machine learning in Python — scikit-learn 0.20.3 documentation," n.d.; "The Python Tutorial — Python 3.7.2 documentation," n.d.) were a source of help in the writing the Python codes.

3.7.1 Multilayer Perceptron (MLP)

As mentioned earlier, the train set was used to train the model, the validation set for selecting the hyperparameters of the model and the test set was used to evaluate the performance of the models on unobserved data. The datasets were scaled between 0 and 1 before feeding them into the MLP model. Later, the model outputs were rescaled back to their original scale.

The various hyperparameters of the MLP model that had to be tuned were the number of prior time step to be fed as input, number of hidden layers, number of neurons in each layer, choice of the optimizer, choice of activation function, learning rate, number of epochs, and batch size. The hyperparameters of the model were selected by manual hyperparameter testing. The model was trained from the training set with different combinations of the values of the hyperparameters and the predictions were made on the validation set for each combination to evaluate the performance of the model and the combination which gave the least mean absolute error (MAE) was selected as the best model. Furthermore, the best model was used to make a prediction on the test set, which was used for reporting the results from the best model. This procedure was repeated for all the three datasets – complete set, weekday set, and weekend set of the three bridges. Additionally, the

computation time was noted, root mean square error (RMSE) was computed to evaluate the performance of the selected model on the test set and graphs were plotted between actual and predicted delay to further analyze the model results.

Additionally, dropout regularization was applied to the last hidden layers to avoid overfitting of the model. The dropout rate was arbitrarily chosen as 20%.

3.7.2 Convolutional Neural Network (CNN)

The CNN model was developed in a similar manner as MLP, which included using training set, validation set, and test set for developing the model, feature scaling, applying dropout, hyperparameter testing, computing performance measures, and plotting graphs.

The hyperparameters of the CNN model were the number of prior time step to be fed as input, number of convolutional layers, filter size, kernel size, max pooling layer size, number of hidden layers, number of neurons in each layer, choice of optimizer, choice of activation function, learning rate, number of epochs, and batch size. These hyperparameters were selected by manual hyperparameter tuning. The loss function was set as a mean absolute error (MAE).

3.7.3 Long Short-Term Memory Recurrent Neural Networks (LSTM RNN)

The LSTM RNN model was also developed in a similar manner as MLP and CNN, this included using training set, validation set, and test set for developing the model, feature scaling, applying dropout, hyperparameter testing, computing performance measures, and plotting graphs.

The hyperparameters of the LSTM RNN model were the number of prior time step to be fed as input, number of LSTM layers, number of hidden layers, number of neurons in each layer, choice of the optimizer, choice of activation function, learning rate, number of epochs, and batch size.

The hyperparameters were selected by manual hyperparameter tuning. The loss function was set as MAE.

3.7.4 Gated Recurrent Unit Recurrent Neural Network (GRU RNN)

The GRU RNN model was developed in a similar way as LSTM RNN, this included using training set, validation set, and test set for developing the model, feature scaling, applying dropout, hyperparameter testing, setting MAE as loss function, computing performance measures, and plotting graphs.

The hyperparameters of the LSTM RNN model were the number of prior time step to be fed as input, number of GRU layers, number of hidden layers, number of neurons in each layer, choice of the optimizer, choice of activation function, learning rate, number of epochs, batch size, and a loss function. The hyperparameters were selected by manual hyperparameter tuning.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Data analysis and findings

The main aim of this research is to predict the traffic delay time series in the future. However, before beginning to develop deep learning models for time series prediction, it seemed rational to analyze the delay data which was obtained from the three border crossings – Peace Bridge, Lewiston-Queenston Bridge, and Rainbow Bridge. This section provides the results of the data analysis conducted on the car delay data to get some insights into it.

4.1.1 Effect of bridges

This study aims to help the travelers entering the U.S. from Canada, in making the right choice of the border crossing bridge to avoid delay. Thus, it was important to evaluate the delay characteristics at each bridge. So, firstly, the comparative analysis was done across the three bridges by comparing the average car traffic delay throughout the day in the month of May 2018, as shown in figure 4.1. By analyzing the graph in figure 4.1, it is clear that the pattern of delay variation at each bridge is different from others. On average, there is always some delay at the Peace Bridge throughout the day, whereas, the delay at Lewiston-Queenston Bridge and Rainbow Bridge reduces to zero or is negligible during the night hours.

Another inference from figure 4.1 is related to the delay peaks for each bridge. Peace Bridge has mainly two peaks of average delay, one at around 03:00 and another around 17:00. On the other

hand, Lewiston-Queenston and Rainbow Bridge have just a single peak of average delay at around 11:00 and 12:00 respectively.

Lastly, it can also be observed that the maximum average delay is highest for Peace Bridge and lowest for the Rainbow Bridge.

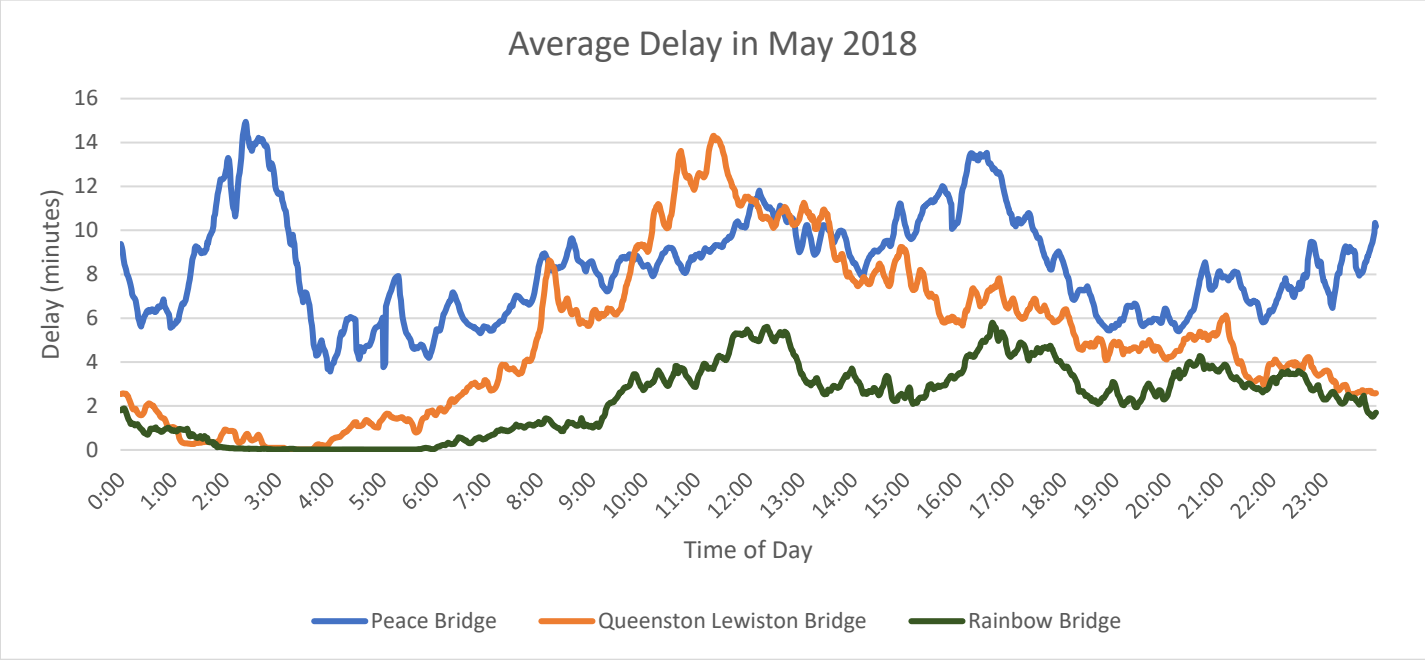


Figure 4. 1: Comparing the average U.S. bound car traffic delay in a day at the bridges during May 2018

To deepen our understanding of the delays at each bridge, the magnitude of delay were compared among the three bridges for May 2018, as shown in figure 4.2. From figure 4.2, it can be seen that mostly the car traffic delay at the Rainbow bridge were less than 5 minutes in duration, whereas at Peace Bridge, there is a significant percentage of a longer delay. Long delays are most common at Peace Bridge, followed by Lewiston-Queenston Bridge, and least common at Rainbow Bridge.

To further to deepen our understanding about the distribution of delay duration at bridges, the duration of the delay was compared among the bridges for weekdays from 07:00 to 22:00, as

shown in figure 4.3. From figure 4.1 shows that Lewiston-Queenston Bridge has the biggest share of delays equal to or longer than 30 minutes. The delay at Rainbow bridge is again mostly less than 5 minutes, but that on Peace Bridge and Lewiston-Queenston Bridge is more uniformly distributed.

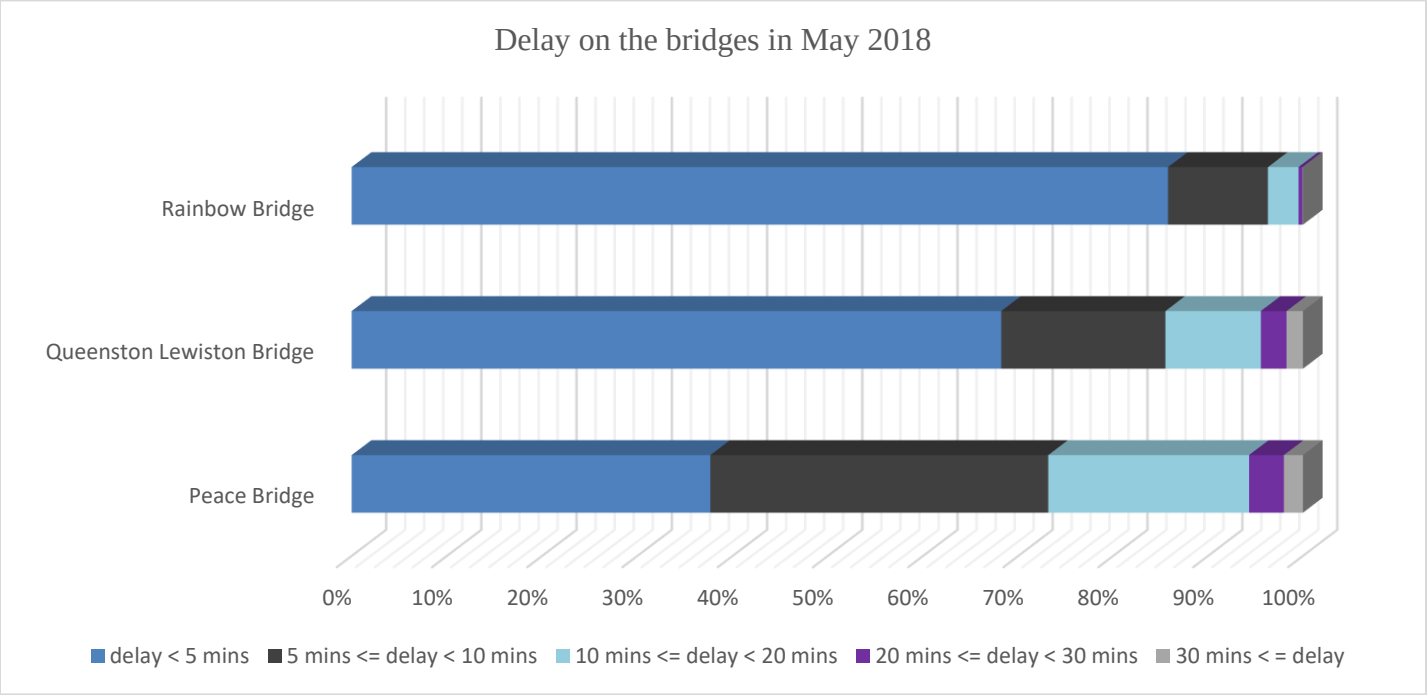


Figure 4. 2: Comparing the U.S. bound car traffic delay across the three bridges in May 2018

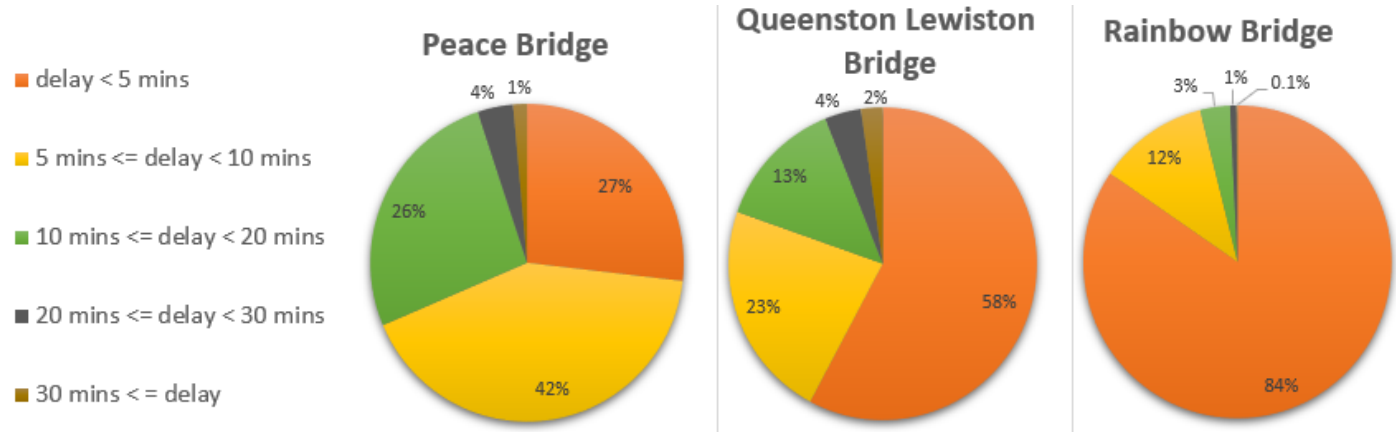


Figure 4. 3: Comparing U.S. bound car traffic delay across the bridges during the weekdays of May 2018 from 7:00 to 22:00

4.1.2 Effect of weekdays and weekends

This study also aims to evaluate the performance of deep learning methods for categorized (weekdays and weekends) and uncategorized data (complete set). The effect of weekdays and weekends on U.S. bound car traffic delay was evaluated by plotting the average delay on the bridges for a complete set, weekdays, and weekends, as shown in figure 4.4.

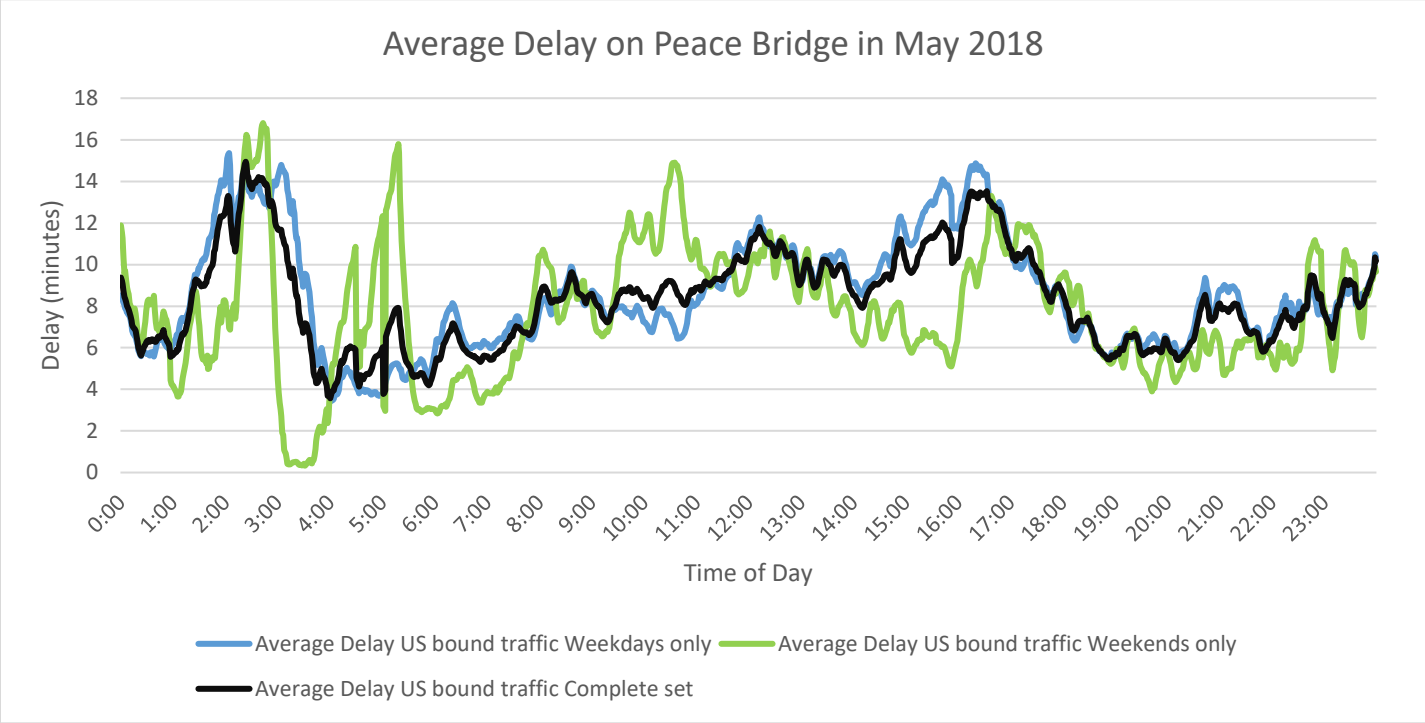


Figure 4. 4: Comparing the average U.S. bound car traffic delay at Peace Bridge during weekdays, weekends, and complete set in May 2018

From figure 4.4, it can be seen that broadly, weekend and weekday set follow the same pattern, but the weekend set is more abrupt. The maximum average car traffic delay was experienced in the weekend set. The weekdays set closely follows the complete set which involves both weekdays and weekends. Similar results were obtained when this kind of graph was plotted for Lewiston-Queenston Bridge and Rainbow Bridge, as shown in figure A.1 and A.2 respectively.

4.1.3 Effect of months/seasons

Another factor that was evaluated to find its effect on delay was different months or seasons. Figure 4.5 shows the variation in U.S. bound car traffic delay experienced on Peace Bridge in different months corresponding to different seasons – April (spring), June (summer), October (fall), and December (winter).

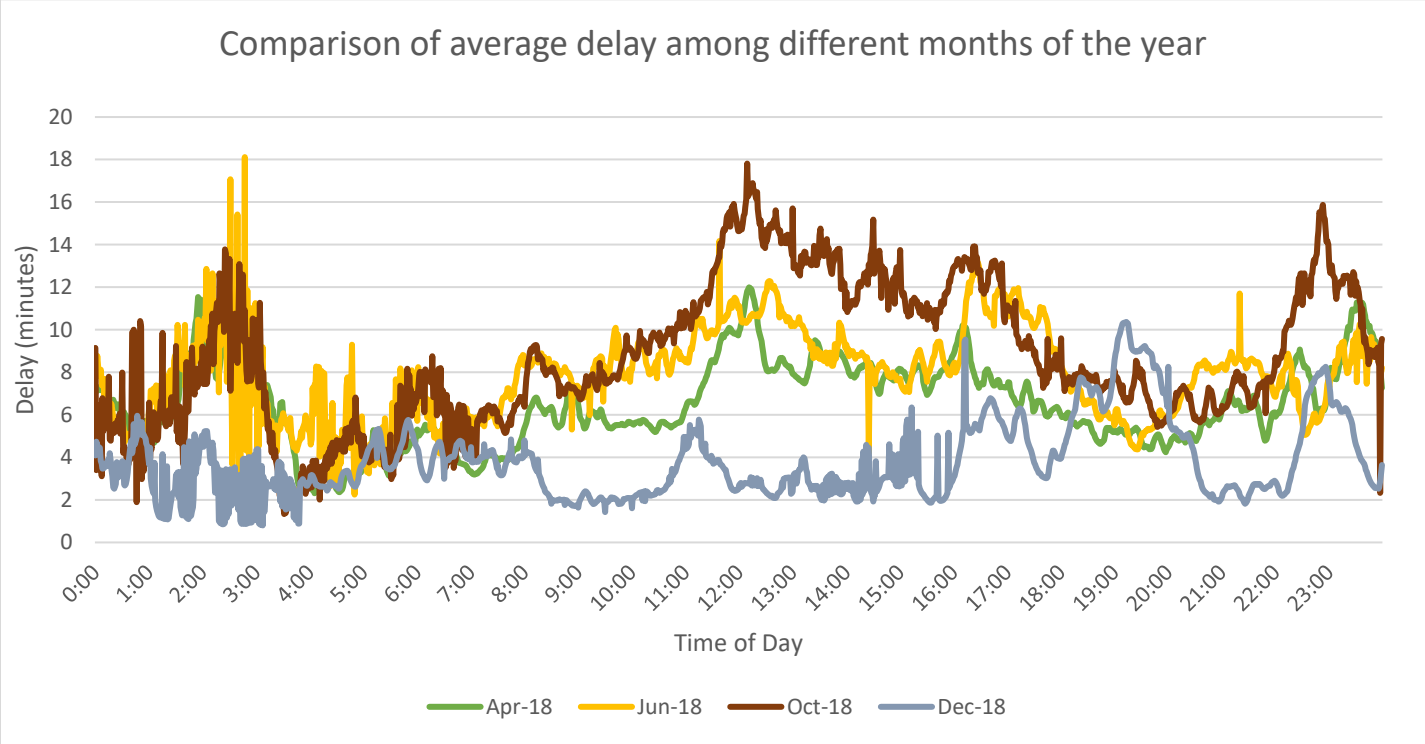


Figure 4. 5: Comparing the average U.S. bound car traffic delay at Peace Bridge across different months

As seen in figure 4.5, the average car traffic delay in October is generally the longest among the other months compared here, while that in December was mostly the shortest. However, broadly, the pattern of delay variation has not changed with the months.

4.1.4 Effect of holidays

The effect of holidays on the car traffic delay at the bridges was analyzed by plotting the delay at the three bridges on the Fourth of July, 2018 as shown in figure 4.6. From figure 4.6, it can be seen that the Peace Bridge is most affected by the holiday. The delay at Peace Bridge is maximum during 02:00 and 03:00, when the delay has even crossed 60 minutes duration for some time. Lewiston-Queenston Bridge and Rainbow Bridge don't seem to be affected by the holiday.

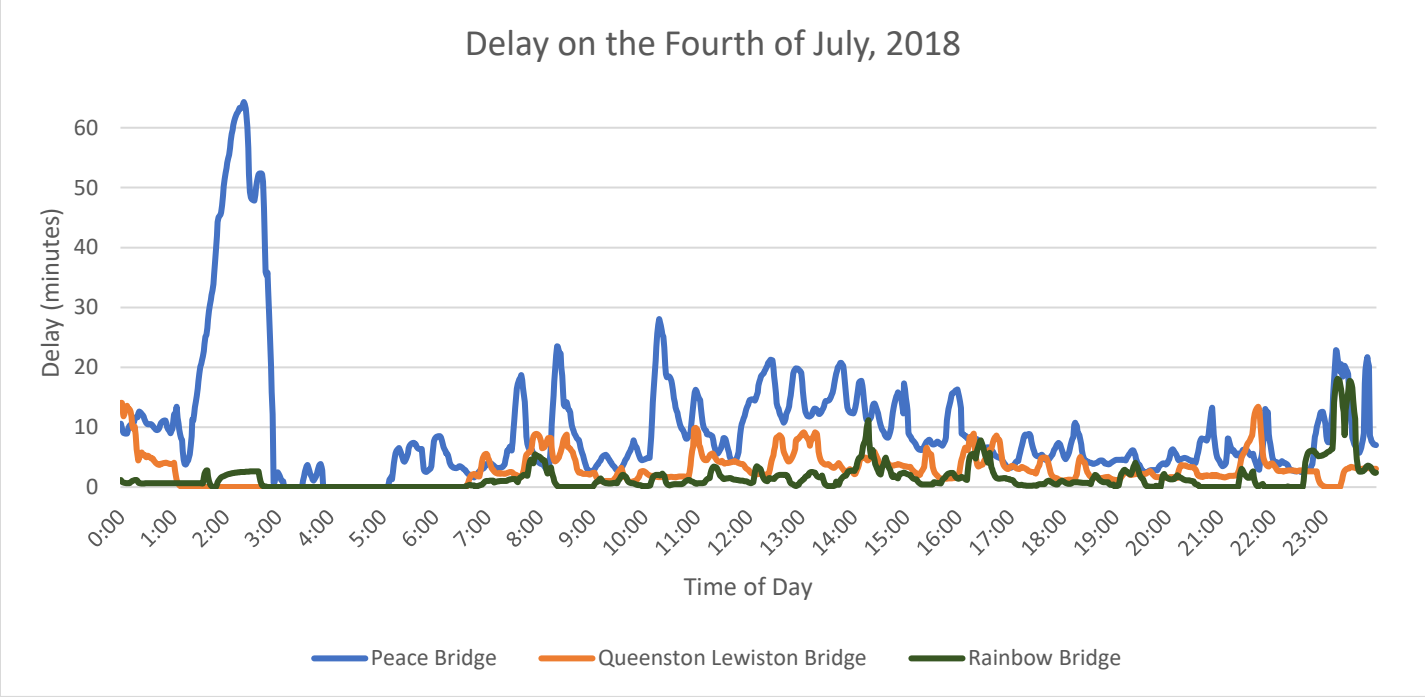


Figure 4. 6: Comparing the U.S. bound car traffic delay at the bridges on the Fourth of July, 2018

4.1.5 Effect of direction of travel

Figure 4.7 shows the variation in car traffic delay across the U.S. bound and Canada bound traffic. It can be observed in figure 4.7 that the delay pattern in the two directions of travel followed a different pattern. Except for a few hours, the average delay in U.S. bound traffic direction is longer than that for Canada bound traffic. The highest average delay for the U.S. bound car traffic is

around 03:00, whereas that for Canada bound traffic is around 19:00. Additionally, unlike the average traffic delay for U.S. bound car traffic, the average traffic delay for Canada bound car traffic is negligible during the night hours.

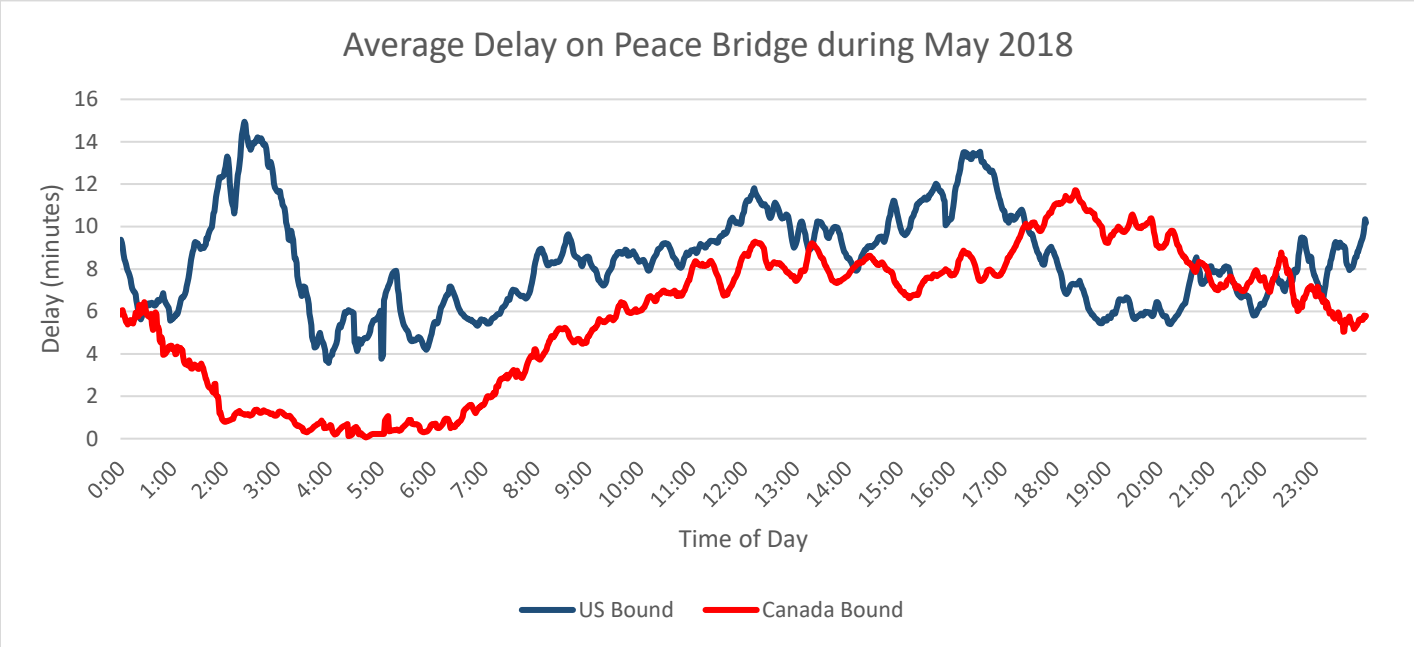


Figure 4. 7: Comparing delay across the U.S. bound and Canada bound traffic on Peace Bridge during May 2018

4.2 Model Results

This section provides the results of the deep learning models developed to predict the U.S. bound car traffic delay at the three border crossings between the U.S. and Canada – Peace Bridge (PB), Lewiston-Queenston Bridge (QL), and Rainbow Bridge (RB). The prediction performance of models was measured by Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and R squared (R^2). The results obtained from each of the deep learning models are presented in detail in the subsequent sections.

4.2.1 Multilayer Perceptron (MLP)

The MLP model was developed to predict the U.S. bound car traffic delay for the next 5 minutes, 10 minutes, 15 minutes, … up to 60 minutes. However, for simplicity, the model results of only next 5, 15, 30, 45, and 60 are shown in table 4.1.

Prediction for	Measure	PB			QL			RB		
		Complete set	Week days	Week ends	Complete set	Week days	Week ends	Complete set	Week days	Week ends
5 minutes	MAE	1.14	1.38	0.92	1.02	1.03	1.08	1.09	1.27	1.62
	RMSE	2.94	3.49	3.28	2.12	2.17	1.96	1.7	1.6	2.34
	R^2	0.95	0.89	0.96	0.95	0.94	0.96	0.93	0.81	0.83
15 minutes	MAE	2.1	2.16	2.01	1.93	1.87	2.19	2.02	2.15	2.57
	RMSE	7.02	6.6	8.5	7.65	7.01	9.21	3.37	3.04	4.31
	R^2	0.79	0.7	0.86	0.83	0.8	0.86	0.79	0.66	0.73
30 minutes	MAE	2.77	2.71	2.72	2.64	2.44	3.06	2.68	2.76	3.35
	RMSE	4	3.96	4.29	4.76	4.65	5.24	5.22	5.77	6.54
	R^2	0.64	0.52	0.75	0.71	0.68	0.74	0.65	0.5	0.6
45 minutes	MAE	3.08	2.98	3.15	3	2.75	3.59	3.06	3.03	3.83
	RMSE	4.52	4.4	5.11	5.48	5.34	6.16	6	6.17	7.25
	R^2	0.54	0.41	0.64	0.62	0.57	0.64	0.54	0.43	0.5
60 minutes	MAE	3.11	3.08	3.49	3.25	2.96	3.99	3.34	3.24	4.24
	RMSE	4.85	4.68	5.75	6.1	5.82	6.96	6.61	6.64	8.01
	R^2	0.48	0.33	0.54	0.53	0.49	0.54	0.44	0.34	0.39
Computation time (s)		165.93	92.74	90.34	114.15	57.28	29.74	62.63	65.73	29.57

Table 4. 1: MLP model result

From table 4.1, it can be seen that MAEs of weekdays are mostly lesser than that of their corresponding complete set. However, for 5 minute prediction, MAEs of the complete set are always lesser than that of their corresponding weekdays. Additionally, it can be noted that except for a few predictions, the MAEs of weekends are the highest among the three datasets of the same bridge and prediction horizon.

From table 4.1, it can also be observed that with the increase of prediction horizon, MAE has always increased while R squared has always decreased for all datasets and all bridges. This decrease in prediction performance with an increase in prediction horizon is reflected in figures 4.8 and 4.9. Figure 4.8 and 4.9 compares the actual delay with a predicted delay for the next 5 and 30 minutes respectively. It can be seen that the model is able to very well follow the actual delay in case of next 5 minute delay prediction, while its performance has decreased for next 30 minutes delay prediction especially for high peaks arriving abruptly. Unlike MAE and R squared, RMSE has not constantly increased or decreased with the increase in the prediction horizon.

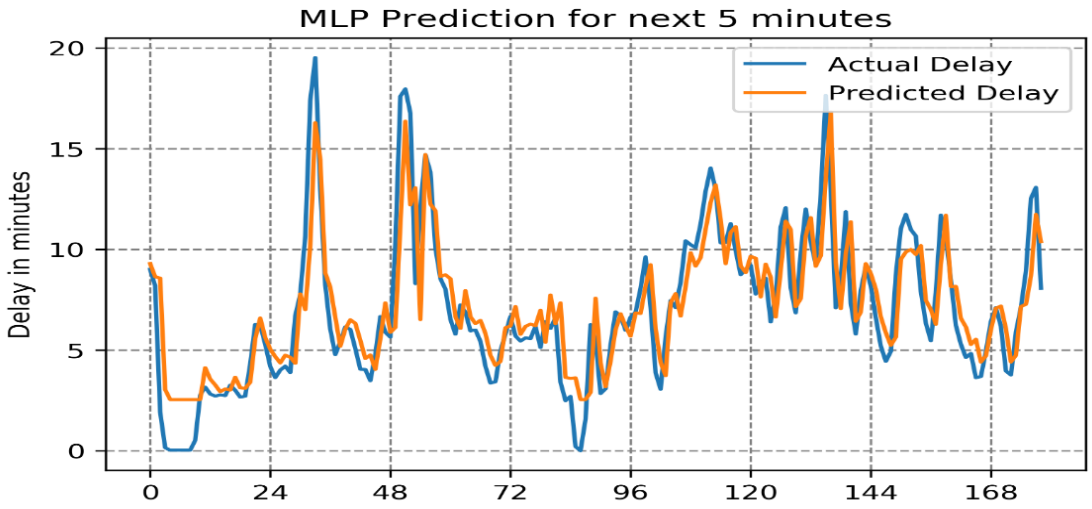


Figure 4. 8: Comparing the actual U.S. bound traffic delay with 5 minutes ahead prediction of delay by the MLP model at Peace Bridge for a sample of 180 data points

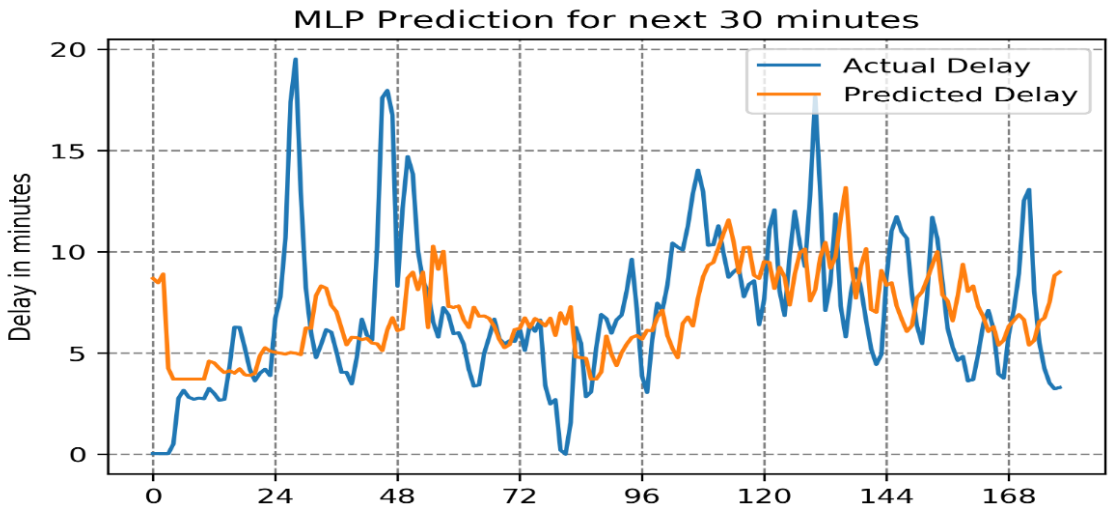


Figure 4. 9: Comparing the actual U.S. bound traffic delay with 30 minutes ahead prediction of delay by the MLP model at Peace Bridge for a sample of 180 data points

Further, the computation time came out to be the highest for the complete set of PB. The computation time of the MLP models for any dataset has not gone beyond 3 minutes. Some additional graphs showing the prediction performance of the MLP models are shown in figures C.1-C.8.

4.2.2 Convolutional Neural Networks (CNN)

Table 4.2 shows the model results of the CNN model in predicting delay for the next 5, 15, 30. 45, and 60 minutes. From table 4.2, it is clear that the MAEs of weekdays are mostly lesser their corresponding complete set, whereas, weekends always had the highest MAE among the three datasets of the same bridge and same prediction horizon.

Prediction for	Measure	PB			QL			RB		
		Complete set	Weekdays	Weekends	Complete set	Weekdays	Weekends	Complete set	Weekdays	Weekends
5 minutes	MAE	0.91	0.98	1.09	0.93	1.01	1.38	1.14	0.98	1.22
	RMSE	2.86	2.79	3.85	1.96	2.15	2.55	1.8	1.58	2.02
	R^2	0.95	0.94	0.96	0.96	0.94	0.94	0.93	0.94	0.95
15 minutes	MAE	1.97	2.1	2.11	1.93	1.85	2.29	2.07	1.93	2.19
	RMSE	6.82	6.22	8.09	7.59	6.98	9.46	3.35	2.96	4.11
	R^2	0.8	0.73	0.85	0.83	0.8	0.84	0.79	0.78	0.85
30 minutes	MAE	2.57	2.64	2.72	2.58	2.42	3.04	2.7	2.51	3.03
	RMSE	3.9	3.79	4.3	4.79	4.67	5.42	5.21	4.85	5.43
	R^2	0.66	0.56	0.75	0.71	0.67	0.72	0.65	0.65	0.72
45 minutes	MAE	2.9	2.97	3.19	2.91	2.72	3.52	3.02	2.85	3.51
	RMSE	4.45	4.21	5.03	5.51	5.36	6.37	5.84	5.6	6.39
	R^2	0.56	0.46	0.65	0.61	0.57	0.62	0.57	0.53	0.61
60 minutes	MAE	3.12	3.03	3.5	3.16	2.94	3.89	3.27	3.04	3.89
	RMSE	4.77	4.42	5.53	6.09	5.85	7.09	6.34	6.05	7.17
	R^2	0.49	0.4	0.58	0.53	0.49	0.52	0.49	0.45	0.51
Computation time (s)		266.41	268.21	124.11	479.51	264.37	382.8	358.26	484.07	332.62

Table 4. 2: CNN model results

Similar to MLPs, the MAEs have always increased and R squared has always decreased with the increase in prediction horizon. However, RMSE doesn't seem to constantly increase or decrease with the prediction horizon. The longest computation time for any CNN model developed in this study was 484.07 seconds, while the minimum was 124.11 seconds.

Figure 4.10 compares the actual and predicted delay for the next 30 minutes by the CNN model. It can be seen from figure 4.10 that the prediction made by this model is able to follow the trend of the time series. Additional graphs showing the performance of CNN models are presented as figures C.9-C.16.

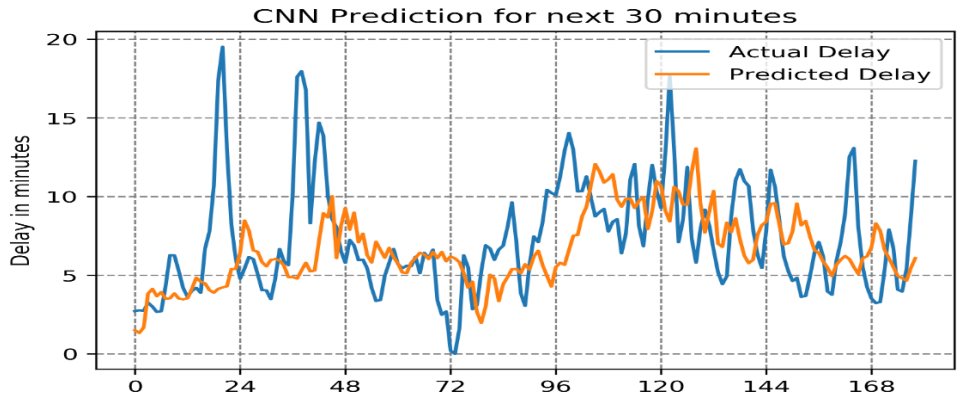


Figure 4. 10: Comparing the actual U.S. bound traffic delay with 30 minutes ahead prediction of delay by CNN model at Peace Bridge for a sample of 180 data points

4.2.3 Long Short-Term Memory Recurrent Neural Networks (LSTM RNN)

The model results of LSTM RNN model are tabulated in table 4.3. From table 4.3, it can be noted that the MAE of weekdays are always the least among its corresponding complete set and weekends, while the weekends are often the highest. With the increase in the prediction horizon, the MAEs have increased, while R squared has reduced. The computation time of LSTM RNN models developed in this study varied from 50.41 seconds to 184.07 seconds.

Figure 4.11 shows the graphical representation of actual delay and delay predicted by LSTM RNN model. The graph shows the ability of the model to tune its prediction with the changes in the prior time steps of the time series. More such graphs regarding the prediction by LSTM RNN are shown in figure C.17-C.24.

Prediction for	Measure	PB			QL			RB		
		Complete set	Weekdays	Weekends	Complete set	Weekdays	Weekends	Complete set	Weekdays	Weekends
5 minutes	MAE	1.02	0.91	0.95	1.14	0.86	1.34	1.27	0.92	1.51
	RMSE	3.33	3.16	3.2	2.45	1.94	2.58	1.9	1.5	2.37
	R^2	0.93	0.93	0.97	0.94	0.95	0.95	0.86	0.94	0.87
15 minutes	MAE	2.01	1.93	1.98	1.99	1.84	2.27	2.14	1.87	2.43
	RMSE	6.95	6.28	8.06	7.37	6.86	8.85	3.31	3	4.06
	R^2	0.78	0.72	0.86	0.82	0.81	0.85	0.74	0.79	0.76
30 minutes	MAE	2.51	2.44	2.66	2.6	2.43	3.04	2.73	2.47	3.16
	RMSE	3.96	3.73	4.35	4.92	4.58	5.3	5.35	4.75	6.17
	R^2	0.65	0.57	0.74	0.69	0.69	0.74	0.64	0.66	0.64
45 minutes	MAE	2.88	2.74	3.07	2.94	2.72	3.53	3.09	2.78	3.57
	RMSE	4.46	4.17	5.02	5.51	5.26	6.21	6.01	5.4	6.85
	R^2	0.56	0.47	0.65	0.61	0.58	0.64	0.54	0.56	0.56
60 minutes	MAE	3.12	3.01	3.39	3.14	2.93	3.92	3.34	2.99	3.94
	RMSE	4.85	4.46	5.51	6.07	5.73	6.98	6.5	5.85	7.52
	R^2	0.47	0.39	0.58	0.53	0.51	0.54	0.46	0.49	0.46
Computation time (s)		184.07	136.06	67.39	167.89	171.03	62.85	166.93	150.95	50.41

Table 4. 3: LSTM RNN model results

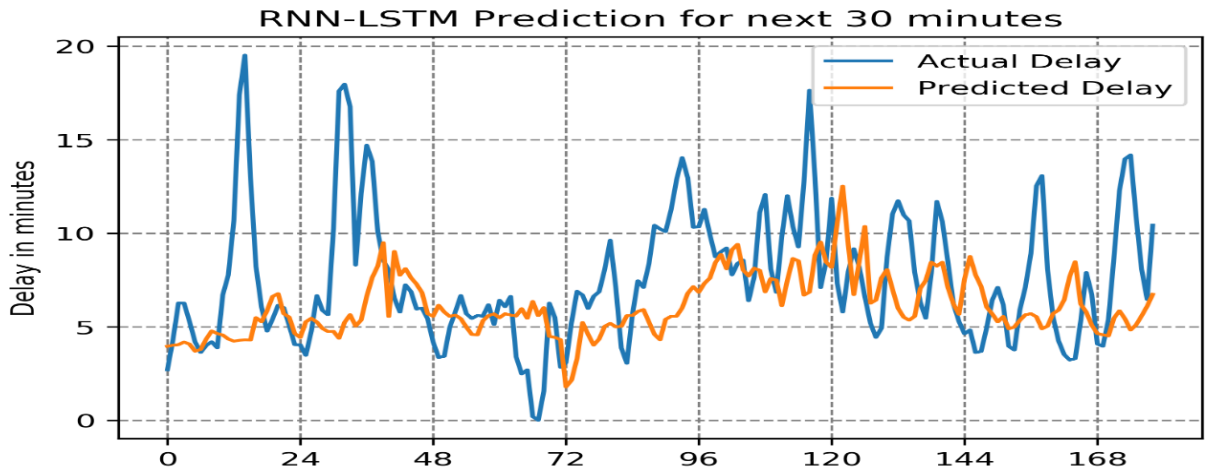


Figure 4. 11: Comparing the actual U.S. bound traffic delay with 30 minutes ahead prediction of delay by LSTM RNN model at Peace Bridge for a sample of 180 data points

4.2.4 Gated Recurrent Unit Recurrent Neural Networks (GRU RNN)

Table 4.4 shows the prediction results for the next 5, 15, 30, and 45 minutes. From table 4.4, it can be seen that the MAEs of weekdays are usually lesser than that of their corresponding complete set. Similar to the MLP, CNN, and LSTM RNN, the MAE and R squared of GRU RNN increases and reduces respectively with the increase in prediction horizon. The computation time of GRU RNNs developed for this study ranges between 30.09 seconds and 173.5 seconds.

Prediction for	Measure	PB			QL			RB		
		Complete set	Weekdays	Weekends	Complete set	Weekdays	Weekends	Complete set	Weekdays	Weekends
5 minutes	MAE	1.21	0.99	1.13	0.96	1.05	1.23	1.37	1.27	1.16
	RMSE	2.88	2.8	5.22	2.04	2.48	2.36	2.01	1.66	1.93
	R^2	0.94	0.94	0.95	0.96	0.93	0.96	0.89	0.84	0.96
15 minutes	MAE	2.11	2.08	2.02	1.93	1.92	2.17	2.26	2.13	2.19
	RMSE	7.01	6.26	7.97	7.48	6.68	8.93	3.52	2.96	4.13
	R^2	0.78	0.73	0.86	0.83	0.79	0.86	0.74	0.68	0.85
30 minutes	MAE	2.6	2.58	2.68	2.58	2.45	2.92	2.8	2.8	3.01
	RMSE	4.03	3.78	4.34	4.7	4.76	5.08	5.38	5.91	5.37
	R^2	0.64	0.56	0.74	0.72	0.66	0.76	0.63	0.48	0.73
45 minutes	MAE	2.96	2.79	3.17	2.93	2.74	3.41	3.18	3.13	3.47
	RMSE	4.53	4.15	5.05	5.42	5.44	5.95	6.19	6.58	6.31
	R^2	0.54	0.47	0.65	0.63	0.56	0.67	0.51	0.35	0.62
60 minutes	MAE	3.15	3.12	3.49	3.17	2.92	3.82	3.44	3.31	3.86
	RMSE	4.9	4.47	5.57	5.97	5.89	6.74	6.68	6.87	7.12
	R^2	0.46	0.39	0.57	0.55	0.48	0.57	0.43	0.29	0.52
Computation time (s)		173.5	111.96	41.47	207.15	111.63	30.75	165.34	114.08	30.09

Table 4. 4: GRU RNN model results

Figure 4.12 shows the graphical representation of actual delay and delay predicted by GRU RNN model for the next 30 minutes. It can be seen from figure 4.12 that predicted mostly follows the predicted delay but misses out abrupt peaks. Additional graphs of prediction results by GRU RNN model are shown in figure C.25-C32.

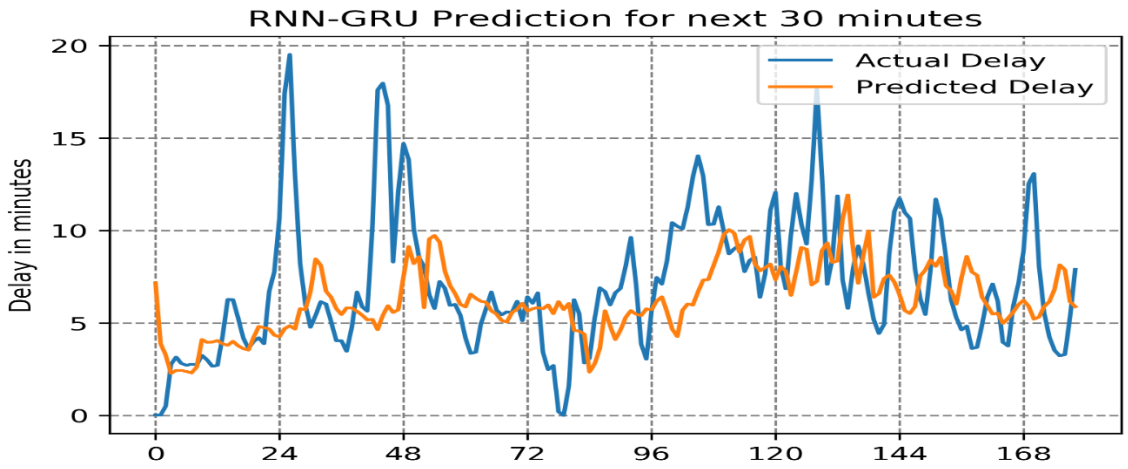


Figure 4. 12: Comparing the actual U.S. bound traffic delay with 30 minutes ahead prediction of delay by GRU RNN model at Peace Bridge for a sample of 180 data points

4.3 Model Comparison

To find the most suitable deep learning technique in predicting short term delay, the model results of the four techniques used in this study were compared for all three bridges and for different prediction horizons. Figure 4.13 and 4.14 compares the four techniques based on the MAEs of the predicted delay at each bridge for 30 minutes ahead and 60 minutes ahead prediction respectively.

From figure 4.13, it could be noticed that the lowest MAE for next 30 minutes prediction at PB and RB are obtained from LSTM RNN and MLP models respectively. Whereas, both CNN and GRU RNN models give the lowest MAE for 30 minutes prediction in QL. Further, from figure 4.14 it can be seen that the lowest MAE for the next 60 minutes prediction at PB, QL, and RB is given by MLP, LSTM RNN, and CNN model respectively. Clearly, there is no absolute winner among these techniques.

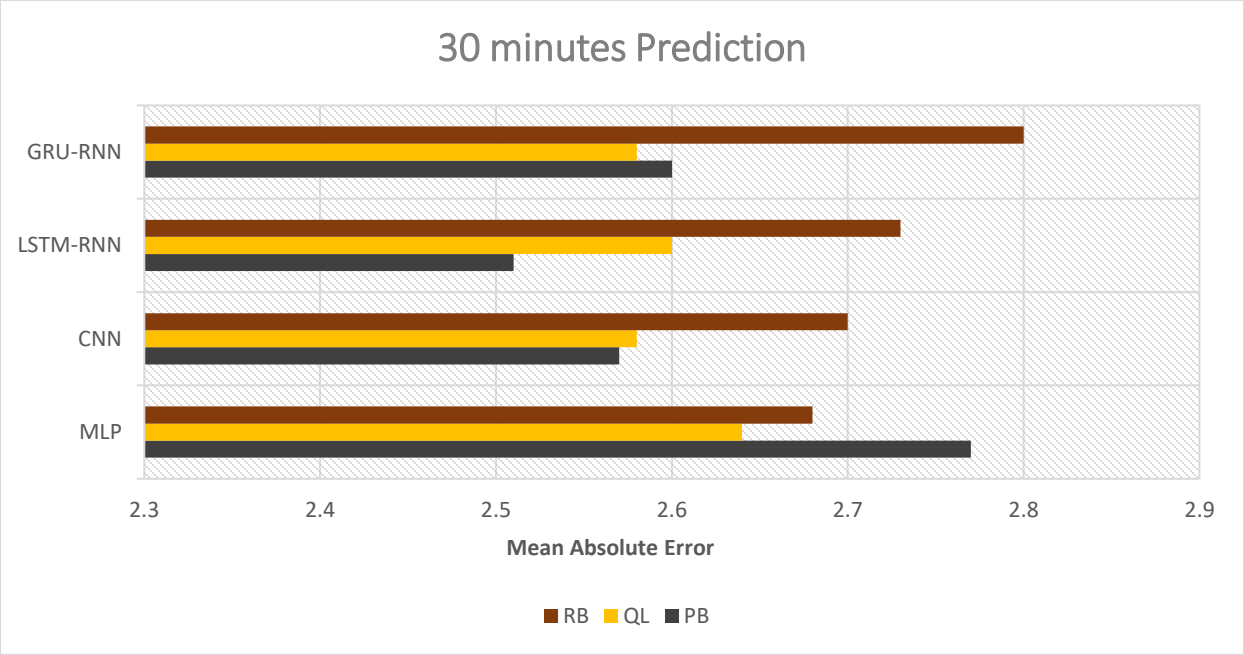


Figure 4. 13: Comparing MAE of delay prediction for next 30 minutes by MLP, CNN, LSTM-RNN, and GRU-RNN at PB, QL, and RB

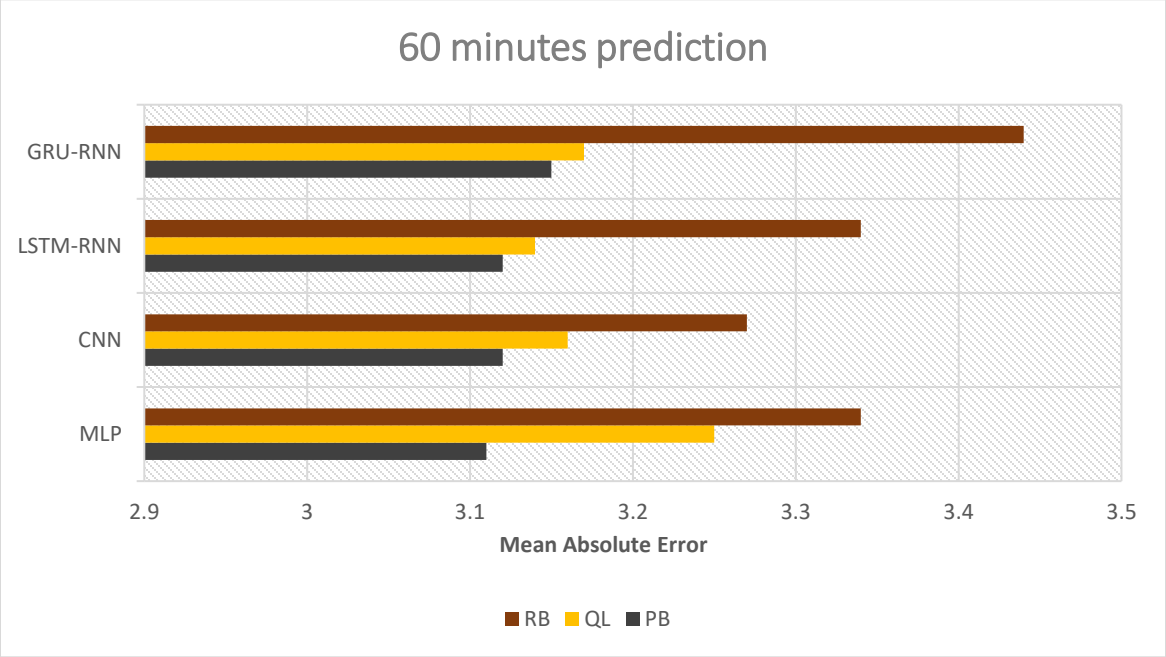


Figure 4. 14: Comparing MAE of delay prediction for the next 60 minutes by MLP, CNN, LSTM-RNN, and GRU-RNN at PB, QL, and RB

As mentioned earlier, the delay at prior time steps that were fed into the model as input. The number of prior time steps that were fed into the models was considered as one of the hyperparameters, which was selected using manual hyperparameter tuning. In this way, each model takes a different number of prior time steps as input. Table 4.5 shows the number of prior time steps that were fed in each model. It must be noted here that each time step is an average delay for the last 5 minutes. Hence, for instance, 12-time steps imply 12 X 5 = 60 minutes, i.e., delays for the last 60 minutes.

Border Crossing	Model	MLP	CNN	LSTM-RNN	GRU-RNN
Peace Bridge	Complete set	16	24	30	18
	Weekdays	16	18	24	18
	Weekends	24	16	18	48
Lewiston-Queenston Bridge	Complete set	24	18	60	36
	Weekdays	16	18	18	48
	Weekends	36	30	48	48
Rainbow Bridge	Complete set	16	30	48	36
	Weekdays	16	18	18	24
	Weekends	36	18	18	24

Table 4. 5: Number of prior time steps used as input for different models

4.4 Effect of data classification on model results

As previously mentioned, the traffic delay time series dataset was classified into weekdays and weekends and thus, three different types of models were developed using the three datasets, which were: complete set, weekdays, and weekends. This section presents the results of the analysis done to investigate the effect of data classification on model results.

The model performance of the complete set models may be compared with weekdays and weekends models by simply comparing the MAEs in their prediction. However, this might not

give the most accurate results as it involves oversampling. This is because the MAEs obtained from the complete set models take into account both weekdays and weekends data. To evaluate the effect of data classification on the performance of deep learning models in a way that avoids the problem of oversampling, the models were run again and the MAEs in the prediction of a specific interval of weekday datapoints and weekend data points by the complete set model were compared with the MAEs in the prediction of the same data points by the weekdays and weekends model respectively. The length of this interval of data points used for the comparison was selected as 288 as it represents the observations of 24 hours ((24*60)/5 =288). Tables 4.6 shows the comparison of the MAEs in the prediction of weekday datapoints and weekend data points by the complete set models with the MAEs in the prediction of the same data points by the weekdays and weekends model respectively for 30 minutes ahead U.S. bound traffic delay prediction at PB.

From table 4.6, it can be seen that the MAEs of the weekday datapoints and weekend data points may be lesser, equal to higher than that obtained from the weekdays model and weekends model respectively. To further analyze the effect of data classification, the same comparison was made for predictions 60 minutes in the future, as shown in table 4.7.

Models	Complete set		Weekdays	Weekends
	Weekday data points	Weekend data points		
MLP	2.25	3.25	2.25	3.44
CNN	2.27	3.38	2.36	3.28
LSTM-RNN	2.15	3.29	2.19	3.39
GRU-RNN	2.13	3.39	2.2	3.21

Table 4. 6: Comparison of MAEs of specific data points in the prediction of delay 30 minutes in future at PB by different models

Models	Complete set		Weekdays	Weekends
	Weekday data points	Weekend data points		
MLP	2.33	3.96	2.38	4.13
CNN	2.5	4.1	2.56	4.05
LSTM-RNN	2.45	3.91	2.48	4
GRU-RNN	2.3	4.01	2.52	4.06

Table 4. 7: Comparison of MAEs of specific data points in the prediction of delay 60 minutes in the future at PB by different models

From table 4.7, it was observed that mostly the MAEs of the weekday data points and weekend data points of complete set were lesser than the MAEs of weekdays model and weekends model respectively. However, there are exceptions to this. In all, it doesn't seem that the data classification increases the predictive accuracy of the models.

4.5. Discussion

To my knowledge, it is the first time that (1) Bluetooth Data from the three Niagara Frontier Border Crossings are used for predicting the border crossing delay, and (2) Deep Learning techniques like CNN, LSTM-RNN, and GRU-RNN are used for predicting delay at the border crossings. The models developed in this study predict delay for next 5, 10, 15, … and so on up to 60 minutes into the future. This information can guide the travelers in selecting the border crossing based on the delay situation. Travelers might have the tendency to choose the border crossing with the least delay and this would help in the uniform distribution of traffic across the crossings. Lin et al. (Lin, Wang, Sadek, et al., 2014) also emphasized on the necessity of predicting the future border crossing delay, which can be helpful for the border crossing authorities in determining the needed staffing level, and also in routing the border-destined traffic intelligently.

The delay prediction process used in this study does not include any data cleaning, the delays are just aggregated to 5 minutes and are fed to the models as an input. This makes the model development and prediction process very straight forward and easy.

Unlike some of the previous studies (A. M. Khan, 2010; Lin F. B. & Lin M. W., 2001), the delay is predicted by models that were developed (trained, validated, and tested) using the field data collected by the Bluetooth readers. Also for the delay analysis part, some of the previous studies (Zhang & Lin, 2017; Zhang et al., 2017) which analyzed the delay at Niagara Frontier Border Crossings lacked the availability of delay data collected by the Bluetooth readers from the Rainbow Bridge. However, the current study analyses the delay using the data collected from all three border crossings. This might have helped in making this research more realistic and appropriate for real-world applications.

The current study differs in many ways from the previous studies which aimed to predict the delays at the border crossings. Lin and Lin (Lin F. B. & Lin M. W., 2001) proposed a delay model which was based on various factors like vehicle processing capacity of a toll gate or inspection gate, volume/capacity ratio, number of available gates, etc. However, the models developed in the current study can predict delay by just using previous delays. Khan (A. M. Khan, 2010) and Moniruzzaman et. al. (Moniruzzaman et al., 2016) have developed ANN for predicting the crossing time, whereas the current study extended these works by using deep learning techniques like CNN, LSTM-RNN, and GRU-RNN which have not been used for this purpose earlier. Moniruzzaman et. al. (Moniruzzaman et al., 2016) forecasted the crossing times for trucks at the Ambassador Bridge border crossing through the ANN model and the training of the model relied on lags of crossing time, truck volume on the bridge, hours of day, and day of week. In contrast, the current study predicted the delays for the passenger cars on the Niagara Frontier Border

Crossings. Another difference is that the training of models in the current study was based purely on previous delays at the crossings. Having just one variable might limit the prediction accuracy of the models but makes the model easy to develop and its easier to obtain the input data. Additionally, it was felt that the information of time of day is already taken into account by the model as the previous delays are fed as the input. Lastly, rather than just focusing on one border crossing as in the case of Moniruzzaman et. al. (Moniruzzaman et al., 2016), the current study focuses on predicting delay at three border crossings which are located in the same region. This clearly broadens the scope of the current study as the idea is not just to develop models which can predict delay at any border crossing but also to guide the travelers in making decision about selecting the crossing that is most appropriate to them, which in turn should help in increasing the efficiency of the borders.

In this study, the delays are predicted directly using the delay data collected from the Bluetooth readers. This can be seen as an improvement over the stepwise border crossing delay prediction model suggested in some of the past studies (Lin, Wang, & Sadek, 2014; Lin, Wang, Sadek, et al., 2014), in which first the future volumes were predicted using past volumes and then using the predicted future volumes, future delay were forecasted. This simplicity in the procedure is also reflected in the prediction accuracy of the current study. The mean absolute difference for 15 minutes ahead forecast of delay at Peace Bridge by this study was just 1.97 minutes (by complete set CNN model), while that obtained by Lin et. al. (Lin, Wang, Sadek, et al., 2014) was about 6.6 minutes. This suggests a higher prediction accuracy of the current study. Not just the improvement in prediction accuracy, the computation time of the models developed in this study were drastically shorter than that required by Lin et. al. (Lin, Wang, & Sadek, 2014).

By conducting delay data analysis, it was found that the delay pattern at the three Niagara Frontier Bridges differed from each other. This result was obtained by Zhang et. al. (Zhang et al., 2017). Such results increase the need for techniques that can predict the future delay so that the travelers can make an informed decision of selecting the border crossing based on the predicted future delay situation across the three. Such a technique is developed in this study using deep learning.

An interesting observation when analyzing the average U.S. bound car traffic delay at the Peace Bridge was that the highest peak delay was during the night hours, around 03:00. There is no clear idea about why there is so much delay at such a time. A possible reason for this could be that there may be fewer number of lanes open during the night hours.

The results from this study suggest high-level accuracy of the deep learning techniques in predicting future traffic delays at the border crossings, with MAEs less than 3.5 minutes in predicting delays for up to 60 minutes into the future by complete set models. Previous studies (Dalto et al., 2015; Fu et al., 2016; Koprinska et al., 2018; Ma et al., 2015) have also supported the superior prediction performance of some of the techniques used in this study. However, prediction-wise, no one deep learning technique emerged as a clear winner among others in predicting the delays. The best deep learning technique that gives the most accurate results changed with the border crossing and the prediction horizon. Although, mostly the prediction accuracies obtained from each of the technique were pretty close to each other.

The computation time for each of the model was noted and it was found that the models took just a few minutes to run, with some model taking lesser than 1 minute time. This computation time included both the time taken in training and predicting. Further, it was also observed that the prediction accuracy of the models mostly decreases with the increase in the prediction horizon.

The hyperparameters of the deep learning models were selected manually by observing the change in the performance of the models on the validation set. The set of hyperparameters that gave the least mean absolute error (MAE) was finalized. However, it was found that the MAEs did not change significantly by tuning the hyperparameters. This suggests that the models are robust.

It was found that the average delay during the weekdays and weekends follow the same pattern but the weekend set was much more abrupt. This study also compared the performance of models trained on the whole data set (called 'complete set' models), versus models that were developed separately for weekdays and for weekends, to examine the effect of data classification on the models' predictive accuracy. The MAEs of the weekday model were mostly the least and that of the weekend models were mostly the highest among the three kinds of the models. Hence, classifying the modeling dataset then developing different models doesn't seem to necessarily improve the predictive performance of the models. There can be various reasons for this. A possible reason could be that the continuity of the time series data was broken when the whole data was classified into weekdays and weekends, and this might have had an adverse effect on the predictive performance of the weekday and weekend models. Another reason could be having Friday included in the weekday model and not the weekend model, although the delay characteristics on Friday might be closer to that on weekends.

The delays were also found to be affected by the choice by of bridges, weekdays and weekends, months/seasons, and direction of travel on traffic delay. However, in the current study, the delays are predicted just by using past delays as input which might have restricted the performance of the models. Hence, the predictive accuracy of the models can be improved by adding some other input variables as well.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Thesis Summary

Traffic delays at the United States-Canada border crossings have adverse effects on the economy as well as the environment. There have been some studies in the past which aimed to analyze the delay at the Niagara Frontier Border Crossings (Zhang & Lin, 2017; Zhang et al., 2017) and others which predicted the future delay/crossing time at the border crossings (A. M. Khan, 2010; Lin F. B. & Lin M. W., 2001; Lin, Wang, & Sadek, 2014; Lin, Wang, Sadek, et al., 2014; Moniruzzaman et al., 2016). This study aims to extend this knowledge. In this study, the predict passenger cars' traffic delays at the three Niagara Frontier Border Crossings, namely the Peace Bridge, the Lewiston-Queenston Bridge, and the Rainbow Bridge for the next 60 minutes into the future, using border wait time data collected by Bluetooth readers recently installed at the crossings. To my knowledge, it is the first time that (1) Bluetooth Data from the three Niagara Frontier Border Crossings are used for predicting the border crossing delay, and (2) Deep Learning techniques like CNN, LSTM-RNN, and GRU-RNN are used for predicting delay at the border crossings

The delay data were analyzed to assess the influence of bridges, weekdays and weekends, months/seasons, holidays, and direction of travel on traffic delay. After which, traffic delays were predicted using four deep learning techniques, namely Multilayer Perceptron (MLP), Convolutional Neural Networks (CNN), Long Short-Term Memory Recurrent Neural Networks (LSTM-RNN), and Gated Recurrent Unit Recurrent Neural Networks (GRU-RNN). The hyperparameters of the models were selected through manual hyperparametric tuning.

The results suggest high-level accuracy of the deep learning techniques in predicting future traffic delays at the border crossings, with MAEs less than 3.5 minutes in predicting delays for up to 60 minutes into the future. Although prediction-wise, no one deep learning technique emerged as a clear winner among others in predicting the delays.

As the delays are predicted directly using the delay data collected from the Bluetooth readers in this study, it can be seen as an improvement over the stepwise border crossing delay prediction model suggested by Lin et. al. (Lin, Wang, Sadek, et al., 2014). This simplicity in the procedure is also reflected in the predictive accuracy of the models. The mean absolute difference for 15 minutes ahead forecast of delay at Peace Bridge by this study was just 1.97 minutes (by complete set CNN model), while that obtained by Lin et. al. (Lin, Wang, Sadek, et al., 2014) was about 6.6 minutes. Not just the improvement in prediction accuracy, the computation time of the models developed in this study were drastically shorter than that required by Lin et. al. (Lin, Wang, & Sadek, 2014).

The study also compared the performance of models trained on the whole data set (called 'complete set' models), versus models that were developed separately for weekdays and for weekends, to examine the effect of data classification on the models' predictive accuracy. For this separate modeling dataset was created for the complete set models, weekday models, and weekend models and then separate models were developed. However, there is no clear evidence that classifying data improves the prediction performance of the models.

The models developed in this study aims to guide travelers in making an informed decision about selecting the border crossing based on the predicted future delay. As the travelers might have the tendency to choose the border crossing with the least delay, the traffic on the crossings could get

uniformly distributed over the three bridges. This should help in increasing the efficiency of the borders.

5.2 Border Crossing Delay Prediction

This study takes border crossing delay prediction to the next step. The delay data collected by the Bluetooth readers placed at the three Niagara Frontier Border Crossing were used for developing the models to predict the future delay. Hence, the both the training and testing of the models were done using field data, instead of simulated data. Using these models, delays can be forecasted up to 60 minutes into the future at the three border crossings. The predictions made by the models have high predictive accuracy and took less computation time.

5.3 Deep Learning Methods

It was found that some of the deep learning techniques have never been used for predicting delay at the border crossings. Hence, after reviewing relevant literature, some deep learning techniques were selected for this purpose, these were Multilayer Perceptron (MLP), Convolutional Neural Networks (CNN), Long Short-Term Memory Recurrent Neural Networks (LSTM-RNN), and Gated Recurrent Unit Recurrent Neural Networks (GRU-RNN). The deep learning techniques have given high prediction accuracies. However, no one deep learning techniques emerged as a clear winner among the all the other techniques used.

5.4 Limitations and Future Work

The current study focuses on predicting future delay using four deep learning techniques, namely MLP, CNN, LSTM-RNN, and GRU-RNN. This task can also be performed by using some other deep learning techniques so as to explore the prediction performance of those techniques. This

research can also be extended by making predictions using statistical models and comparing their performance with the techniques employed in the current study. This can help in conducting comparative analysis among the deep learning techniques and statistical models.

In the current study, predictions are made using just previous delay as input to the deep learning models. The prediction performance can be improved by adding more input variables like car volume, number of open lanes, weather information, holidays, traffic accidents, etc.

The delay prediction models can be turned into an online model where the delay information could be fed automatically and continuously as input into the model.

Lastly, such future delay predicting models can also be developed for Canada bound traffic over the Niagara Frontier Border Crossings. Additionally, this work can be extended to other border crossings as well.

APPENDIX A

FURTHER DATA ANALYSES

This section provides further data analysis conducted for this study.

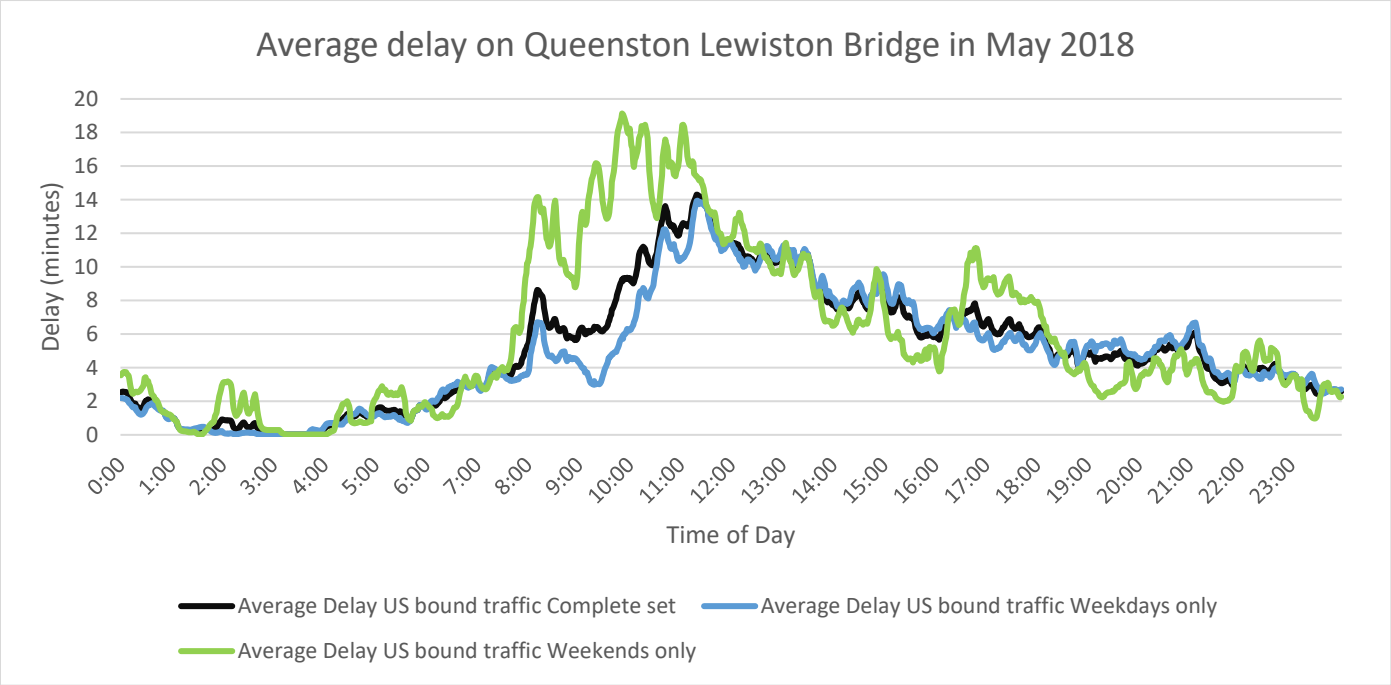


Figure A. 1: Comparing the average U.S. bound car traffic delay at Queenston Lewiston Bridge during weekdays, weekends, and complete set in May 2018

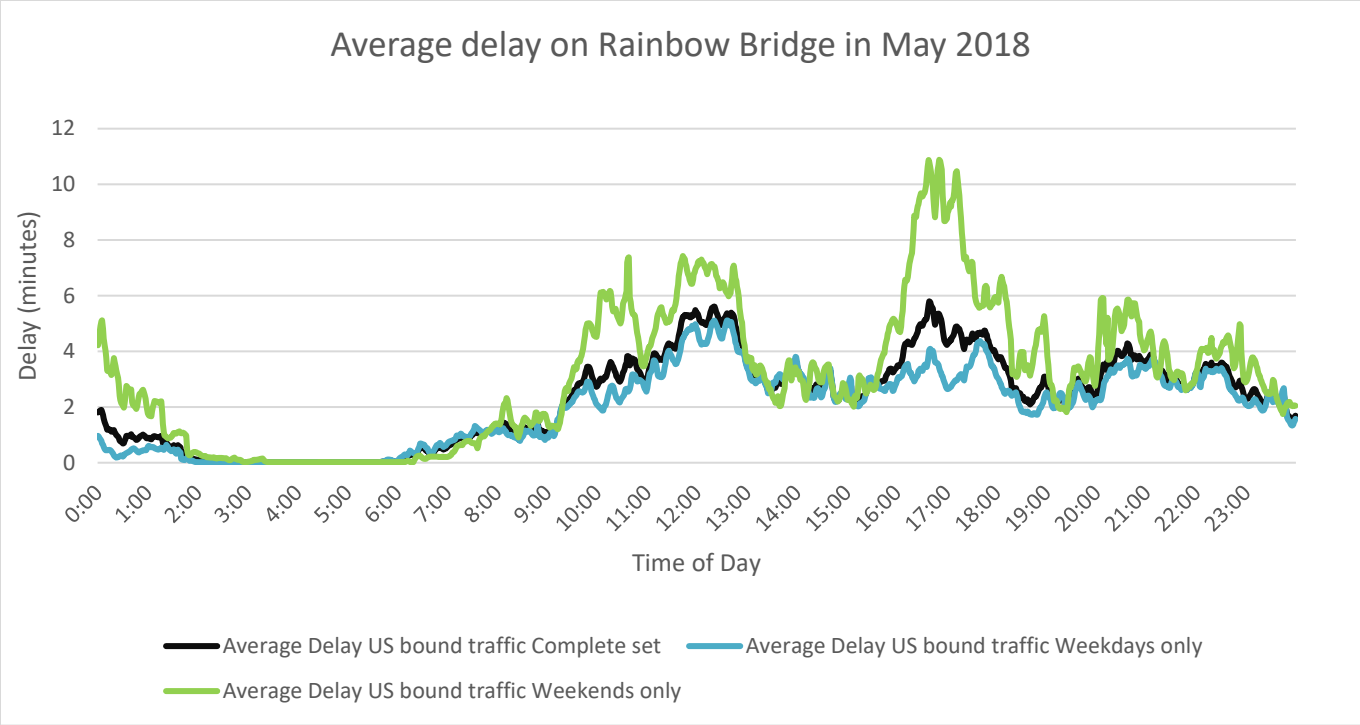


Figure A. 2: Comparing the average U.S. bound car traffic delay at Rainbow Bridge during weekdays, weekends, and complete set in May 2018

APPENDIX B

PYTHON CODES

This section provides the python codes used for developing deep learning models in this study. The python codes used in this study for developing and comparing the deep learning models are inspired by the codes presented by (Brownlee, 2016, 2018; Pal & Prakash, 2017). Additionally, (Ardit, 2017; "Customizing Ticks | Python Data Science Handbook," n.d.; "Home - Keras Documentation," n.d.; "Matplotlib: Python plotting — Matplotlib 3.0.3 documentation," n.d.; "scikit-learn: machine learning in Python — scikit-learn 0.20.3 documentation," n.d.; "The Python Tutorial — Python 3.7.2 documentation," n.d.) were a source of help in the writing the Python codes.

B.1 MLP model python codes

Following are the python code for predicting delay in future using MLP model:

```
import time

time1 = time.time()

import numpy

import matplotlib.pyplot as plt

import pandas as pd

import math
```

```
from keras.models import Sequential

from keras.layers import Dense

from keras.layers import Dropout

from keras.layers import LSTM

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean_squared_error

numpy.random.seed(5)

from keras import optimizers

dataload = pd.read_csv(r'C:\Users\HOME\.spyder-py3\PBUSU10m.csv')

datanew = dataload.values

datanew = datanew.astype('float32')

def datamodified(datanew, timesteps):

        dataX, dataY = [], []

        for i in range(len(datanew)-timesteps-11):

                a = datanew[i:(i+timesteps), 0]

                dataX.append(a)

                dataY.append(datanew[(i+timesteps):(i+timesteps+12), 0])
```

```
        return numpy.array(dataX), numpy.array(dataY)

scaler = MinMaxScaler(feature_range=(0, 1))

datanorm = scaler.fit_transform(datanew)

trainlen = int(len(datanorm) * 0.60)

cvlen = int(len(datanorm) * 0.20)

testlen = len(datanorm) - trainlen - cvlen

train  = datanorm[0:trainlen,:]

cv = datanorm[trainlen:(trainlen+cvlen),:]

test = datanorm[(trainlen+cvlen):len(datanorm),:]

timesteps = 16

trainX, trainY = datamodified(train, timesteps)

cvX, cvY = datamodified(cv, timesteps)

testX, testY = datamodified(test, timesteps)

model = Sequential()

model.add(Dense(30, activation='relu', input_dim=timesteps))

model.add(Dense(40, activation='relu'))

model.add(Dense(20, activation='relu'))
```

```
model.add(Dense(15, activation='relu'))

model.add(Dense(12, activation='relu'))

model.add(Dropout(0.2))

model.add(Dense(12)) #output layer with 3 neurons

model.compile(loss='mean_absolute_error', optimizer='Adam')

model.fit(trainX, trainY, epochs=30, batch_size=20, verbose=2)

print(model.summary())

trainPredict = model.predict(trainX)

cvPredict = model.predict(cvX)

testPredict = model.predict(testX)

trainPredict = scaler.inverse_transform(trainPredict)

cvPredict = scaler.inverse_transform(cvPredict)

testPredict = scaler.inverse_transform(testPredict)

trainY = scaler.inverse_transform(trainY)

cvY = scaler.inverse_transform(cvY)

testY = scaler.inverse_transform(testY)

trainrmse = math.sqrt(mean_squared_error(trainY[:,0], trainPredict[:,0]))
```

```
print('Train rmse 5 min: %.2f' % (trainrmse))

testrmse = math.sqrt(mean_squared_error(trainY[:,2], trainPredict[:,2]))

print('Train rmse 15 min: %.2f' % (testrmse))

trainrmse = math.sqrt(mean_squared_error(trainY[:,5], trainPredict[:,5]))

print('Train rmse 30 min: %.2f' % (trainrmse))

trainrmse = math.sqrt(mean_squared_error(trainY[:,8], trainPredict[:,8]))

print('Train rmse 45 min: %.2f' % (trainrmse))

trainrmse = math.sqrt(mean_squared_error(trainY[:,11], trainPredict[:,11]))

print('Train rmse 60 min: %.2f' % (trainrmse))

cvrmse = math.sqrt(mean_squared_error(cvY[:,0], cvPredict[:,0]))

print('cv rmse 5 min: %.2f' % (cvrmse))

cvrmse = math.sqrt(mean_squared_error(cvY[:,2], cvPredict[:,2]))

print('cv rmse 15 min: %.2f' % (cvrmse))

cvrmse = math.sqrt(mean_squared_error(cvY[:,5], cvPredict[:,5]))

print('cv rmse 30 min: %.2f' % (cvrmse))

cvrmse = math.sqrt(mean_squared_error(cvY[:,8], cvPredict[:,8]))

print('cv rmse 45 min: %.2f' % (cvrmse))
```

```
cvrmse = math.sqrt(mean_squared_error(cvY[:,11], cvPredict[:,11]))

print('cv rmse 60 min: %.2f' % (cvrmse))

testrmse = math.sqrt(mean_squared_error(cvY[:,0], cvPredict[:,0]))

print('Test rmse 5 min: %.2f' % (testrmse))

trainScore = math.sqrt(mean_squared_error(testY[:,2], testPredict[:,2]))

print('Test rmse 15 min: %.2f' % (trainrmse))

testrmse = math.sqrt(mean_squared_error(testY[:,5], testPredict[:,5]))

print('Test rmse 30 min: %.2f' % (testrmse))

testrmse = math.sqrt(mean_squared_error(testY[:,8], testPredict[:,8]))

print('Test rmse 45 min: %.2f' % (testrmse))

testrmse = math.sqrt(mean_squared_error(testY[:,11], testPredict[:,11]))

print('Test rmse 60 min: %.2f' % (testrmse))

def mean_absolute_difference(y_true, y_pred):

    y_true, y_pred = numpy.array(y_true), numpy.array(y_pred)

    return numpy.mean(numpy.abs(y_true - y_pred))

md1 = mean_absolute_difference(trainY[:,0], trainPredict[:,0])

md2 = mean_absolute_difference(trainY[:,2], trainPredict[:,2])
```

```
md3 = mean_absolute_difference(trainY[:,5], trainPredict[:,5])

md4 = mean_absolute_difference(trainY[:,8], trainPredict[:,8])

md5 = mean_absolute_difference(trainY[:,11], trainPredict[:,11])

md6 = mean_absolute_difference(cvY[:,0], cvPredict[:,0])

md7 = mean_absolute_difference(cvY[:,2], cvPredict[:,2])

md8 = mean_absolute_difference(cvY[:,5], cvPredict[:,5])

md9 = mean_absolute_difference(cvY[:,8], cvPredict[:,8])

md10 = mean_absolute_difference(cvY[:,11], cvPredict[:,11])

md11 = mean_absolute_difference(testY[:,0], testPredict[:,0])

md12 = mean_absolute_difference(testY[:,2], testPredict[:,2])

md13 = mean_absolute_difference(testY[:,5], testPredict[:,5])

md14 = mean_absolute_difference(testY[:,8], testPredict[:,8])

md15 = mean_absolute_difference(testY[:,11], testPredict[:,11])

print('Train 5 min MD: %.2f' % (md1))

print('Train 15 min MD: %.2f' % (md2))

print('Train 30 min MD: %.2f' % (md3))

print('Train 45 min MD: %.2f' % (md4))
```

```
print('Train 60 min MD: %.2f' % (md5))

print('cv 5 min MD: %.2f' % (md6))

print('cv 15 min MD: %.2f' % (md7))

print('cv 30 min MD: %.2f' % (md8))

print('cv 45 min MD: %.2f' % (md9))

print('cv 60 min MD: %.2f' % (md10))

print('Test 5 min MD: %.2f' % (md11))

print('Test 15 min MD: %.2f' % (md12))

print('Test 30 min MD: %.2f' % (md13))

print('Test 45 min MD: %.2f' % (md14))

print('Test 60 min MD: %.2f' % (md15))

from sklearn import metrics

R25train= metrics.r2_score(trainY[:,0], trainPredict[:,0], sample_weight=None)

R215train= metrics.r2_score(trainY[:,2], trainPredict[:,2], sample_weight=None)

R230train= metrics.r2_score(trainY[:,5], trainPredict[:,5], sample_weight=None)

R245train= metrics.r2_score(trainY[:,8], trainPredict[:,8], sample_weight=None)

R260train= metrics.r2_score(trainY[:,11], trainPredict[:,11], sample_weight=None)
```

```
R25cv= metrics.r2_score(cvY[:,0], cvPredict[:,0], sample_weight=None)

R215cv= metrics.r2_score(cvY[:,2], cvPredict[:,2], sample_weight=None)

R230cv= metrics.r2_score(cvY[:,5], cvPredict[:,5], sample_weight=None)

R245cv= metrics.r2_score(cvY[:,8], cvPredict[:,8], sample_weight=None)

R260cv= metrics.r2_score(cvY[:,11], cvPredict[:,11], sample_weight=None)

R25test= metrics.r2_score(testY[:,0], testPredict[:,0], sample_weight=None)

R215test= metrics.r2_score(testY[:,2], testPredict[:,2], sample_weight=None)

R230test= metrics.r2_score(testY[:,5], testPredict[:,5], sample_weight=None)

R245test= metrics.r2_score(testY[:,8], testPredict[:,8], sample_weight=None)

R260test= metrics.r2_score(testY[:,11], testPredict[:,11], sample_weight=None)

print('Train 5 min R2: %.2f' % (R25train))

print('Train 15 min R2: %.2f' % (R215train))

print('Train 30 min R2: %.2f' % (R230train))

print('Train 45 min R2: %.2f' % (R245train))

print('Train 60 min R2: %.2f' % (R260train))

print('cv 5 min R2: %.2f' % (R25cv))

print('cv 15 min R2: %.2f' % (R215cv))
```

```
print('cv 30 min R2: %.2f' % (R230cv))

print('cv 45 min R2: %.2f' % (R245cv))

print('cv 60 min R2: %.2f' % (R260cv))

print('Test 5 min R2: %.2f' % (R25test))

print('Test 15 min R2: %.2f' % (R215test))

print('Test 30 min R2: %.2f' % (R230test))

print('Test 45 min R2: %.2f' % (R245test))

print('Test 60 min R2: %.2f' % (R260test))

time2 = time.time()

print(time2 - time1)
```

B.2 CNN model python codes

Following are the python code for predicting delay in future using CNN model

```
import time

time1 = time.time()

import numpy

import matplotlib.pyplot as plt

import pandas as pd
```

```
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
numpy.random.seed(5)
from keras import optimizers
dataload = pd.read_csv(r'C:\Users\HOME\.spyder-py3\PBUSU10m.csv')
datanew = dataload.values
datanew = datanew.astype('float32')
def datamodified(datanew, timesteps):
        dataX, dataY = [], []
        for i in range(len(datanew)-timesteps-11):
                a = datanew[i:(i+timesteps), 0]
                dataX.append(a)
```

```
                dataY.append(datanew[(i+timesteps):(i+timesteps+12), 0])
        return numpy.array(dataX), numpy.array(dataY)
scaler = MinMaxScaler(feature_range=(0, 1))
datanorm = scaler.fit_transform(datanew)
trainlen = int(len(datanorm) * 0.60)
cvlen = int(len(datanorm) * 0.20)
testlen = len(datanorm) - trainlen - cvlen
train  = datanorm[0:trainlen,:]
cv = datanorm[trainlen:(trainlen+cvlen),:]
test = datanorm[(trainlen+cvlen):len(datanorm),:]
timesteps = 24
trainX, trainY = datamodified(train, timesteps)
cvX, cvY = datamodified(cv, timesteps)
testX, testY = datamodified(test, timesteps)
trainX = numpy.reshape(trainX, (trainX.shape[0], timesteps, 1))
cvX = numpy.reshape(cvX, (cvX.shape[0], timesteps, 1))
testX = numpy.reshape(testX, (testX.shape[0], timesteps, 1))
```

```
from keras.layers import Flatten

from keras.layers.convolutional import Conv1D

from keras.layers.convolutional import MaxPooling1D

model = Sequential()

model.add(Conv1D(filters=150, kernel_size=12, activation='relu', input_shape=(timesteps, 1)))

model.add(Conv1D(filters=50, kernel_size=4, activation='relu'))

model.add(MaxPooling1D(pool_size=2))

model.add(Flatten())

model.add(Dense(300, activation='relu'))

model.add(Dense(320, activation='relu'))

model.add(Dropout(0.2))

model.add(Dense(12))

model.compile(loss='mean_absolute_error', optimizer='AdaGrad')

model.fit(trainX, trainY, epochs=20, batch_size=60, verbose=2)

print(model.summary())

trainPredict = model.predict(trainX)

cvPredict = model.predict(cvX)
```

```
testPredict = model.predict(testX)

trainPredict = scaler.inverse_transform(trainPredict)

cvPredict = scaler.inverse_transform(cvPredict)

testPredict = scaler.inverse_transform(testPredict)

trainB  = datanew[0:trainlen,:]

cvB = datanew[trainlen:(trainlen+cvlen),:]

testB = datanew[(trainlen+cvlen):len(datanew),:]

trainBX, trainBY = datamodified(trainB, timesteps)

cvBX, cvBY = datamodified(cvB, timesteps)

testBX, testBY = datamodified(testB, timesteps)

trainrmse = math.sqrt(mean_squared_error(trainBY[:,0], trainPredict[:,0]))

print('Train rmse 5 min: %.2f' % (trainrmse))

testrmse = math.sqrt(mean_squared_error(trainBY[:,2], trainPredict[:,2]))

print('Train rmse 15 min: %.2f' % (testrmse))

trainrmse = math.sqrt(mean_squared_error(trainBY[:,5], trainPredict[:,5]))

print('Train rmse 30 min: %.2f' % (trainrmse))

trainrmse = math.sqrt(mean_squared_error(trainBY[:,8], trainPredict[:,8]))
```

```
print('Train rmse 45 min: %.2f' % (trainrmse))

trainrmse = math.sqrt(mean_squared_error(trainBY[:,11], trainPredict[:,11]))

print('Train rmse 60 min: %.2f' % (trainrmse))

cvrmse = math.sqrt(mean_squared_error(cvBY[:,0], cvPredict[:,0]))

print('cv rmse 5 min: %.2f' % (cvrmse))

cvrmse = math.sqrt(mean_squared_error(cvBY[:,2], cvPredict[:,2]))

print('cv rmse 15 min: %.2f' % (cvrmse))

cvrmse = math.sqrt(mean_squared_error(cvBY[:,5], cvPredict[:,5]))

print('cv rmse 30 min: %.2f' % (cvrmse))

cvrmse = math.sqrt(mean_squared_error(cvBY[:,8], cvPredict[:,8]))

print('cv rmse 45 min: %.2f' % (cvrmse))

cvrmse = math.sqrt(mean_squared_error(cvBY[:,11], cvPredict[:,11]))

print('cv rmse 60 min: %.2f' % (cvrmse))

testrmse = math.sqrt(mean_squared_error(cvBY[:,0], cvPredict[:,0]))

print('Test rmse 5 min: %.2f' % (testrmse))

trainScore = math.sqrt(mean_squared_error(testBY[:,2], testPredict[:,2]))

print('Test rmse 15 min: %.2f' % (trainrmse))
```

```
testrmse = math.sqrt(mean_squared_error(testBY[:,5], testPredict[:,5]))

print('Test rmse 30 min: %.2f' % (testrmse))

testrmse = math.sqrt(mean_squared_error(testBY[:,8], testPredict[:,8]))

print('Test rmse 45 min: %.2f' % (testrmse))

testrmse = math.sqrt(mean_squared_error(testBY[:,11], testPredict[:,11]))

print('Test rmse 60 min: %.2f' % (testrmse))

def mean_absolute_difference(y_true, y_pred):

    y_true, y_pred = numpy.array(y_true), numpy.array(y_pred)

    return numpy.mean(numpy.abs(y_true - y_pred))

md1 = mean_absolute_difference(trainBY[:,0], trainPredict[:,0])

md2 = mean_absolute_difference(trainBY[:,2], trainPredict[:,2])

md3 = mean_absolute_difference(trainBY[:,5], trainPredict[:,5])

md4 = mean_absolute_difference(trainBY[:,8], trainPredict[:,8])

md5 = mean_absolute_difference(trainBY[:,11], trainPredict[:,11])

md6 = mean_absolute_difference(cvBY[:,0], cvPredict[:,0])

md7 = mean_absolute_difference(cvBY[:,2], cvPredict[:,2])

md8 = mean_absolute_difference(cvBY[:,5], cvPredict[:,5])
```

```
md9 = mean_absolute_difference(cvBY[:,8], cvPredict[:,8])

md10 = mean_absolute_difference(cvBY[:,11], cvPredict[:,11])

md11 = mean_absolute_difference(testBY[:,0], testPredict[:,0])

md12 = mean_absolute_difference(testBY[:,2], testPredict[:,2])

md13 = mean_absolute_difference(testBY[:,5], testPredict[:,5])

md14 = mean_absolute_difference(testBY[:,8], testPredict[:,8])

md15 = mean_absolute_difference(testBY[:,11], testPredict[:,11])

print('Train 5 min MD: %.2f' % (md1))

print('Train 15 min MD: %.2f' % (md2))

print('Train 30 min MD: %.2f' % (md3))

print('Train 45 min MD: %.2f' % (md4))

print('Train 60 min MD: %.2f' % (md5))

print('cv 5 min MD: %.2f' % (md6))

print('cv 15 min MD: %.2f' % (md7))

print('cv 30 min MD: %.2f' % (md8))

print('cv 45 min MD: %.2f' % (md9))

print('cv 60 min MD: %.2f' % (md10))
```

```
print('Test 5 min MD: %.2f' % (md11))

print('Test 15 min MD: %.2f' % (md12))

print('Test 30 min MD: %.2f' % (md13))

print('Test 45 min MD: %.2f' % (md14))

print('Test 60 min MD: %.2f' % (md15))

from sklearn import metrics

R25train= metrics.r2_score(trainBY[:,0], trainPredict[:,0], sample_weight=None)

R215train= metrics.r2_score(trainBY[:,2], trainPredict[:,2], sample_weight=None)

R230train= metrics.r2_score(trainBY[:,5], trainPredict[:,5], sample_weight=None)

R245train= metrics.r2_score(trainBY[:,8], trainPredict[:,8], sample_weight=None)

R260train= metrics.r2_score(trainBY[:,11], trainPredict[:,11], sample_weight=None)

R25cv= metrics.r2_score(cvBY[:,0], cvPredict[:,0], sample_weight=None)

R215cv= metrics.r2_score(cvBY[:,2], cvPredict[:,2], sample_weight=None)

R230cv= metrics.r2_score(cvBY[:,5], cvPredict[:,5], sample_weight=None)

R245cv= metrics.r2_score(cvBY[:,8], cvPredict[:,8], sample_weight=None)

R260cv= metrics.r2_score(cvBY[:,11], cvPredict[:,11], sample_weight=None)

R25test= metrics.r2_score(testBY[:,0], testPredict[:,0], sample_weight=None)
```

```
R215test= metrics.r2_score(testBY[:,2], testPredict[:,2], sample_weight=None)

R230test= metrics.r2_score(testBY[:,5], testPredict[:,5], sample_weight=None)

R245test= metrics.r2_score(testBY[:,8], testPredict[:,8], sample_weight=None)

R260test= metrics.r2_score(testBY[:,11], testPredict[:,11], sample_weight=None)

print('Train 5 min R2: %.2f' % (R25train))

print('Train 15 min R2: %.2f' % (R215train))

print('Train 30 min R2: %.2f' % (R230train))

print('Train 45 min R2: %.2f' % (R245train))

print('Train 60 min R2: %.2f' % (R260train))

print('cv 5 min R2: %.2f' % (R25cv))

print('cv 15 min R2: %.2f' % (R215cv))

print('cv 30 min R2: %.2f' % (R230cv))

print('cv 45 min R2: %.2f' % (R245cv))

print('cv 60 min R2: %.2f' % (R260cv))

print('Test 5 min R2: %.2f' % (R25test))

print('Test 15 min R2: %.2f' % (R215test))

print('Test 30 min R2: %.2f' % (R230test))
```

```
print('Test 45 min R2: %.2f' % (R245test))

print('Test 60 min R2: %.2f' % (R260test))

time2 = time.time()

print(time2 - time1)
```

B.3 LSTM RNN model python codes

Following are the python code for predicting delay in future using LSTM RNN model:

```
import time

time1 = time.time()

import numpy

import matplotlib.pyplot as plt

import pandas as pd

import math

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import Dropout

from keras.layers import LSTM

from sklearn.preprocessing import MinMaxScaler
```

```
from sklearn.metrics import mean_squared_error

numpy.random.seed(5)

from keras import optimizers

dataload = pd.read_csv(r'C:\Users\HOME\.spyder-py3\PBUSU10m.csv')

datanew = dataload.values

datanew = datanew.astype('float32')

def datamodified(datanew, timesteps):

        dataX, dataY = [], []

        for i in range(len(datanew)-timesteps-11):

                a = datanew[i:(i+timesteps), 0]

                dataX.append(a)

                dataY.append(datanew[(i+timesteps):(i+timesteps+12), 0])

        return numpy.array(dataX), numpy.array(dataY)

scaler = MinMaxScaler(feature_range=(0, 1))

datanorm = scaler.fit_transform(datanew)

trainlen = int(len(datanorm) * 0.60)

cvlen = int(len(datanorm) * 0.20)
```

```
testlen = len(datanorm) - trainlen - cvlen

train  = datanorm[0:trainlen,:]

cv = datanorm[trainlen:(trainlen+cvlen),:]

test = datanorm[(trainlen+cvlen):len(datanorm),:]

timesteps = 30

trainX, trainY = datamodified(train, timesteps)

cvX, cvY = datamodified(cv, timesteps)

testX, testY = datamodified(test, timesteps)

trainX = numpy.reshape(trainX, (trainX.shape[0], 1, timesteps))

cvX = numpy.reshape(cvX, (cvX.shape[0], 1, timesteps))

testX = numpy.reshape(testX, (testX.shape[0], 1, timesteps))

model = Sequential()

model.add(Dense(30,activation='relu')) #a hidden layer with 8 neurons

model.add(LSTM(50, input_shape=(trainX.shape[1], timesteps))) #LSTM layer with 10 neurons

model.add(Dense(25,activation='relu'))

model.add(Dropout(0.2))

model.add(Dense(12)) #output layer with 3 neurons
```

```
model.compile(loss='mean_absolute_error', optimizer='Adam')

model.fit(trainX, trainY, epochs=25, batch_size=30, verbose=2)

print(model.summary())

trainPredict = model.predict(trainX)

cvPredict = model.predict(cvX)

testPredict = model.predict(testX)

trainPredict = scaler.inverse_transform(trainPredict)

cvPredict = scaler.inverse_transform(cvPredict)

testPredict = scaler.inverse_transform(testPredict)

trainB  = datanew[0:trainlen,:]

cvB = datanew[trainlen:(trainlen+cvlen),:]

testB = datanew[(trainlen+cvlen):len(datanew),:]

trainBX, trainBY = datamodified(trainB, timesteps)

cvBX, cvBY = datamodified(cvB, timesteps)

testBX, testBY = datamodified(testB, timesteps)

trainrmse = math.sqrt(mean_squared_error(trainBY[:,0], trainPredict[:,0]))

print('Train rmse 5 min: %.2f' % (trainrmse))
```

```
testrmse = math.sqrt(mean_squared_error(trainBY[:,2], trainPredict[:,2]))

print('Train rmse 15 min: %.2f' % (testrmse))

trainrmse = math.sqrt(mean_squared_error(trainBY[:,5], trainPredict[:,5]))

print('Train rmse 30 min: %.2f' % (trainrmse))

trainrmse = math.sqrt(mean_squared_error(trainBY[:,8], trainPredict[:,8]))

print('Train rmse 45 min: %.2f' % (trainrmse))

trainrmse = math.sqrt(mean_squared_error(trainBY[:,11], trainPredict[:,11]))

print('Train rmse 60 min: %.2f' % (trainrmse))

cvrmse = math.sqrt(mean_squared_error(cvBY[:,0], cvPredict[:,0]))

print('cv rmse 5 min: %.2f' % (cvrmse))

cvrmse = math.sqrt(mean_squared_error(cvBY[:,2], cvPredict[:,2]))

print('cv rmse 15 min: %.2f' % (cvrmse))

cvrmse = math.sqrt(mean_squared_error(cvBY[:,5], cvPredict[:,5]))

print('cv rmse 30 min: %.2f' % (cvrmse))

cvrmse = math.sqrt(mean_squared_error(cvBY[:,8], cvPredict[:,8]))

print('cv rmse 45 min: %.2f' % (cvrmse))

cvrmse = math.sqrt(mean_squared_error(cvBY[:,11], cvPredict[:,11]))
```

```
print('cv rmse 60 min: %.2f' % (cvrmse))

testrmse = math.sqrt(mean_squared_error(cvBY[:,0], cvPredict[:,0]))

print('Test rmse 5 min: %.2f' % (testrmse))

trainScore = math.sqrt(mean_squared_error(testBY[:,2], testPredict[:,2]))

print('Test rmse 15 min: %.2f' % (trainrmse))

testrmse = math.sqrt(mean_squared_error(testBY[:,5], testPredict[:,5]))

print('Test rmse 30 min: %.2f' % (testrmse))

testrmse = math.sqrt(mean_squared_error(testBY[:,8], testPredict[:,8]))

print('Test rmse 45 min: %.2f' % (testrmse))

testrmse = math.sqrt(mean_squared_error(testBY[:,11], testPredict[:,11]))

print('Test rmse 60 min: %.2f' % (testrmse))

def mean_absolute_difference(y_true, y_pred):

    y_true, y_pred = numpy.array(y_true), numpy.array(y_pred)

    return numpy.mean(numpy.abs(y_true - y_pred))

md1 = mean_absolute_difference(trainBY[:,0], trainPredict[:,0])

md2 = mean_absolute_difference(trainBY[:,2], trainPredict[:,2])

md3 = mean_absolute_difference(trainBY[:,5], trainPredict[:,5])
```

```
md4 = mean_absolute_difference(trainBY[:,8], trainPredict[:,8])

md5 = mean_absolute_difference(trainBY[:,11], trainPredict[:,11])

md6 = mean_absolute_difference(cvBY[:,0], cvPredict[:,0])

md7 = mean_absolute_difference(cvBY[:,2], cvPredict[:,2])

md8 = mean_absolute_difference(cvBY[:,5], cvPredict[:,5])

md9 = mean_absolute_difference(cvBY[:,8], cvPredict[:,8])

md10 = mean_absolute_difference(cvBY[:,11], cvPredict[:,11])

md11 = mean_absolute_difference(testBY[:,0], testPredict[:,0])

md12 = mean_absolute_difference(testBY[:,2], testPredict[:,2])

md13 = mean_absolute_difference(testBY[:,5], testPredict[:,5])

md14 = mean_absolute_difference(testBY[:,8], testPredict[:,8])

md15 = mean_absolute_difference(testBY[:,11], testPredict[:,11])

print('Train 5 min MD: %.2f' % (md1))

print('Train 15 min MD: %.2f' % (md2))

print('Train 30 min MD: %.2f' % (md3))

print('Train 45 min MD: %.2f' % (md4))

print('Train 60 min MD: %.2f' % (md5))
```

```
print('cv 5 min MD: %.2f' % (md6))

print('cv 15 min MD: %.2f' % (md7))

print('cv 30 min MD: %.2f' % (md8))

print('cv 45 min MD: %.2f' % (md9))

print('cv 60 min MD: %.2f' % (md10))

print('Test 5 min MD: %.2f' % (md11))

print('Test 15 min MD: %.2f' % (md12))

print('Test 30 min MD: %.2f' % (md13))

print('Test 45 min MD: %.2f' % (md14))

print('Test 60 min MD: %.2f' % (md15))

from sklearn import metrics

R25train= metrics.r2_score(trainBY[:,0], trainPredict[:,0], sample_weight=None)

R215train= metrics.r2_score(trainBY[:,2], trainPredict[:,2], sample_weight=None)

R230train= metrics.r2_score(trainBY[:,5], trainPredict[:,5], sample_weight=None)

R245train= metrics.r2_score(trainBY[:,8], trainPredict[:,8], sample_weight=None)

R260train= metrics.r2_score(trainBY[:,11], trainPredict[:,11], sample_weight=None)

R25cv= metrics.r2_score(cvBY[:,0], cvPredict[:,0], sample_weight=None)
```

```
R215cv= metrics.r2_score(cvBY[:,2], cvPredict[:,2], sample_weight=None)

R230cv= metrics.r2_score(cvBY[:,5], cvPredict[:,5], sample_weight=None)

R245cv= metrics.r2_score(cvBY[:,8], cvPredict[:,8], sample_weight=None)

R260cv= metrics.r2_score(cvBY[:,11], cvPredict[:,11], sample_weight=None)

R25test= metrics.r2_score(testBY[:,0], testPredict[:,0], sample_weight=None)

R215test= metrics.r2_score(testBY[:,2], testPredict[:,2], sample_weight=None)

R230test= metrics.r2_score(testBY[:,5], testPredict[:,5], sample_weight=None)

R245test= metrics.r2_score(testBY[:,8], testPredict[:,8], sample_weight=None)

R260test= metrics.r2_score(testBY[:,11], testPredict[:,11], sample_weight=None)

print('Train 5 min R2: %.2f' % (R25train))

print('Train 15 min R2: %.2f' % (R215train))

print('Train 30 min R2: %.2f' % (R230train))

print('Train 45 min R2: %.2f' % (R245train))

print('Train 60 min R2: %.2f' % (R260train))

print('cv 5 min R2: %.2f' % (R25cv))

print('cv 15 min R2: %.2f' % (R215cv))

print('cv 30 min R2: %.2f' % (R230cv))
```

```
print('cv 45 min R2: %.2f' % (R245cv))

print('cv 60 min R2: %.2f' % (R260cv))

print('Test 5 min R2: %.2f' % (R25test))

print('Test 15 min R2: %.2f' % (R215test))

print('Test 30 min R2: %.2f' % (R230test))

print('Test 45 min R2: %.2f' % (R245test))

print('Test 60 min R2: %.2f' % (R260test))

time2 = time.time()

print(time2 - time1)
```

B.4 GRU RNN model python codes

Following are the python code for predicting delay in future using GRU RNN model:

```
import time

time1 = time.time()

import numpy

import matplotlib.pyplot as plt

import pandas as pd

import math
```

```
from keras.models import Sequential

from keras.layers import Dense

from keras.layers import Dropout

from keras.layers import GRU

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean_squared_error

numpy.random.seed(5)

from keras import optimizers

dataload = pd.read_csv(r'C:\Users\HOME\.spyder-py3\PBUSU10m.csv')

datanew = dataload.values

datanew = datanew.astype('float32')

def datamodified(datanew, timesteps):

        dataX, dataY = [], []

        for i in range(len(datanew)-timesteps-11):

                a = datanew[i:(i+timesteps), 0]

                dataX.append(a)

                dataY.append(datanew[(i+timesteps):(i+timesteps+12), 0])
```

```
        return numpy.array(dataX), numpy.array(dataY)

scaler = MinMaxScaler(feature_range=(0, 1))

datanorm = scaler.fit_transform(datanew)

trainlen = int(len(datanorm) * 0.60)

cvlen = int(len(datanorm) * 0.20)

testlen = len(datanorm) - trainlen - cvlen

train  = datanorm[0:trainlen,:]

cv = datanorm[trainlen:(trainlen+cvlen),:]

test = datanorm[(trainlen+cvlen):len(datanorm),:]

timesteps = 18

trainX, trainY = datamodified(train, timesteps)

cvX, cvY = datamodified(cv, timesteps)

testX, testY = datamodified(test, timesteps)

trainX = numpy.reshape(trainX, (trainX.shape[0], 1, timesteps))

cvX = numpy.reshape(cvX, (cvX.shape[0], 1, timesteps))

testX = numpy.reshape(testX, (testX.shape[0], 1, timesteps))

model = Sequential()
```

```
model.add(Dense(50,activation='relu')) #a hidden layer with 8 neurons

model.add(GRU(100, input_shape=(trainX.shape[1], timesteps))) #LSTM layer with 10 40neurons

model.add(Dense(70,activation='relu'))

model.add(Dense(20,activation='relu'))

model.add(Dropout(0.2))

model.add(Dense(12)) #output layer with 3 neurons

model.compile(loss='mean_absolute_error', optimizer='AdaDelta')

model.fit(trainX, trainY, epochs=30, batch_size=60, verbose=2)

print(model.summary())

trainPredict = model.predict(trainX)

cvPredict = model.predict(cvX)

testPredict = model.predict(testX)

trainPredict = scaler.inverse_transform(trainPredict)

cvPredict = scaler.inverse_transform(cvPredict)

testPredict = scaler.inverse_transform(testPredict)

trainB  = datanew[0:trainlen,:]

cvB = datanew[trainlen:(trainlen+cvlen),:]
```

```
testB = datanew[(trainlen+cvlen):len(datanew),:]

trainBX, trainBY = datamodified(trainB, timesteps)

cvBX, cvBY = datamodified(cvB, timesteps)

testBX, testBY = datamodified(testB, timesteps)

trainrmse = math.sqrt(mean_squared_error(trainBY[:,0], trainPredict[:,0]))

print('Train rmse 5 min: %.2f' % (trainrmse))

testrmse = math.sqrt(mean_squared_error(trainBY[:,2], trainPredict[:,2]))

print('Train rmse 15 min: %.2f' % (testrmse))

trainrmse = math.sqrt(mean_squared_error(trainBY[:,5], trainPredict[:,5]))

print('Train rmse 30 min: %.2f' % (trainrmse))

trainrmse = math.sqrt(mean_squared_error(trainBY[:,8], trainPredict[:,8]))

print('Train rmse 45 min: %.2f' % (trainrmse))

trainrmse = math.sqrt(mean_squared_error(trainBY[:,11], trainPredict[:,11]))

print('Train rmse 60 min: %.2f' % (trainrmse))

cvrmse = math.sqrt(mean_squared_error(cvBY[:,0], cvPredict[:,0]))

print('cv rmse 5 min: %.2f' % (cvrmse))

cvrmse = math.sqrt(mean_squared_error(cvBY[:,2], cvPredict[:,2]))
```

```
print('cv rmse 15 min: %.2f' % (cvrmse))

cvrmse = math.sqrt(mean_squared_error(cvBY[:,5], cvPredict[:,5]))

print('cv rmse 30 min: %.2f' % (cvrmse))

cvrmse = math.sqrt(mean_squared_error(cvBY[:,8], cvPredict[:,8]))

print('cv rmse 45 min: %.2f' % (cvrmse))

cvrmse = math.sqrt(mean_squared_error(cvBY[:,11], cvPredict[:,11]))

print('cv rmse 60 min: %.2f' % (cvrmse))

testrmse = math.sqrt(mean_squared_error(cvBY[:,0], cvPredict[:,0]))

print('Test rmse 5 min: %.2f' % (testrmse))

trainScore = math.sqrt(mean_squared_error(testBY[:,2], testPredict[:,2]))

print('Test rmse 15 min: %.2f' % (trainrmse))

testrmse = math.sqrt(mean_squared_error(testBY[:,5], testPredict[:,5]))

print('Test rmse 30 min: %.2f' % (testrmse))

testrmse = math.sqrt(mean_squared_error(testBY[:,8], testPredict[:,8]))

print('Test rmse 45 min: %.2f' % (testrmse))

testrmse = math.sqrt(mean_squared_error(testBY[:,11], testPredict[:,11]))

print('Test rmse 60 min: %.2f' % (testrmse))
```

```
def mean_absolute_difference(y_true, y_pred):
    y_true, y_pred = numpy.array(y_true), numpy.array(y_pred)
    return numpy.mean(numpy.abs(y_true - y_pred))
md1 = mean_absolute_difference(trainBY[:,0], trainPredict[:,0])
md2 = mean_absolute_difference(trainBY[:,2], trainPredict[:,2])
md3 = mean_absolute_difference(trainBY[:,5], trainPredict[:,5])
md4 = mean_absolute_difference(trainBY[:,8], trainPredict[:,8])
md5 = mean_absolute_difference(trainBY[:,11], trainPredict[:,11])
md6 = mean_absolute_difference(cvBY[:,0], cvPredict[:,0])
md7 = mean_absolute_difference(cvBY[:,2], cvPredict[:,2])
md8 = mean_absolute_difference(cvBY[:,5], cvPredict[:,5])
md9 = mean_absolute_difference(cvBY[:,8], cvPredict[:,8])
md10 = mean_absolute_difference(cvBY[:,11], cvPredict[:,11])
md11 = mean_absolute_difference(testBY[:,0], testPredict[:,0])
md12 = mean_absolute_difference(testBY[:,2], testPredict[:,2])
md13 = mean_absolute_difference(testBY[:,5], testPredict[:,5])
md14 = mean_absolute_difference(testBY[:,8], testPredict[:,8])
```

```
md15 = mean_absolute_difference(testBY[:,11], testPredict[:,11])

print('Train 5 min MD: %.2f' % (md1))

print('Train 15 min MD: %.2f' % (md2))

print('Train 30 min MD: %.2f' % (md3))

print('Train 45 min MD: %.2f' % (md4))

print('Train 60 min MD: %.2f' % (md5))

print('cv 5 min MD: %.2f' % (md6))

print('cv 15 min MD: %.2f' % (md7))

print('cv 30 min MD: %.2f' % (md8))

print('cv 45 min MD: %.2f' % (md9))

print('cv 60 min MD: %.2f' % (md10))

print('Test 5 min MD: %.2f' % (md11))

print('Test 15 min MD: %.2f' % (md12))

print('Test 30 min MD: %.2f' % (md13))

print('Test 45 min MD: %.2f' % (md14))

print('Test 60 min MD: %.2f' % (md15))

from sklearn import metrics
```

```
R25train= metrics.r2_score(trainBY[:,0], trainPredict[:,0], sample_weight=None)

R215train= metrics.r2_score(trainBY[:,2], trainPredict[:,2], sample_weight=None)

R230train= metrics.r2_score(trainBY[:,5], trainPredict[:,5], sample_weight=None)

R245train= metrics.r2_score(trainBY[:,8], trainPredict[:,8], sample_weight=None)

R260train= metrics.r2_score(trainBY[:,11], trainPredict[:,11], sample_weight=None)

R25cv= metrics.r2_score(cvBY[:,0], cvPredict[:,0], sample_weight=None)

R215cv= metrics.r2_score(cvBY[:,2], cvPredict[:,2], sample_weight=None)

R230cv= metrics.r2_score(cvBY[:,5], cvPredict[:,5], sample_weight=None)

R245cv= metrics.r2_score(cvBY[:,8], cvPredict[:,8], sample_weight=None)

R260cv= metrics.r2_score(cvBY[:,11], cvPredict[:,11], sample_weight=None)

R25test= metrics.r2_score(testBY[:,0], testPredict[:,0], sample_weight=None)

R215test= metrics.r2_score(testBY[:,2], testPredict[:,2], sample_weight=None)

R230test= metrics.r2_score(testBY[:,5], testPredict[:,5], sample_weight=None)

R245test= metrics.r2_score(testBY[:,8], testPredict[:,8], sample_weight=None)

R260test= metrics.r2_score(testBY[:,11], testPredict[:,11], sample_weight=None)

print('Train 5 min R2: %.2f' % (R25train))

print('Train 15 min R2: %.2f' % (R215train))
```

```
print('Train 30 min R2: %.2f' % (R230train))

print('Train 45 min R2: %.2f' % (R245train))

print('Train 60 min R2: %.2f' % (R260train))

print('cv 5 min R2: %.2f' % (R25cv))

print('cv 15 min R2: %.2f' % (R215cv))

print('cv 30 min R2: %.2f' % (R230cv))

print('cv 45 min R2: %.2f' % (R245cv))

print('cv 60 min R2: %.2f' % (R260cv))

print('Test 5 min R2: %.2f' % (R25test))

print('Test 15 min R2: %.2f' % (R215test))

print('Test 30 min R2: %.2f' % (R230test))

print('Test 45 min R2: %.2f' % (R245test))

print('Test 60 min R2: %.2f' % (R260test))

time2 = time.time()

print(time2 - time1)
```

APPENDIX C

DETAILED MODELING RESULTS

This section provides detailed modeling results.

C.1 MLP model prediction results

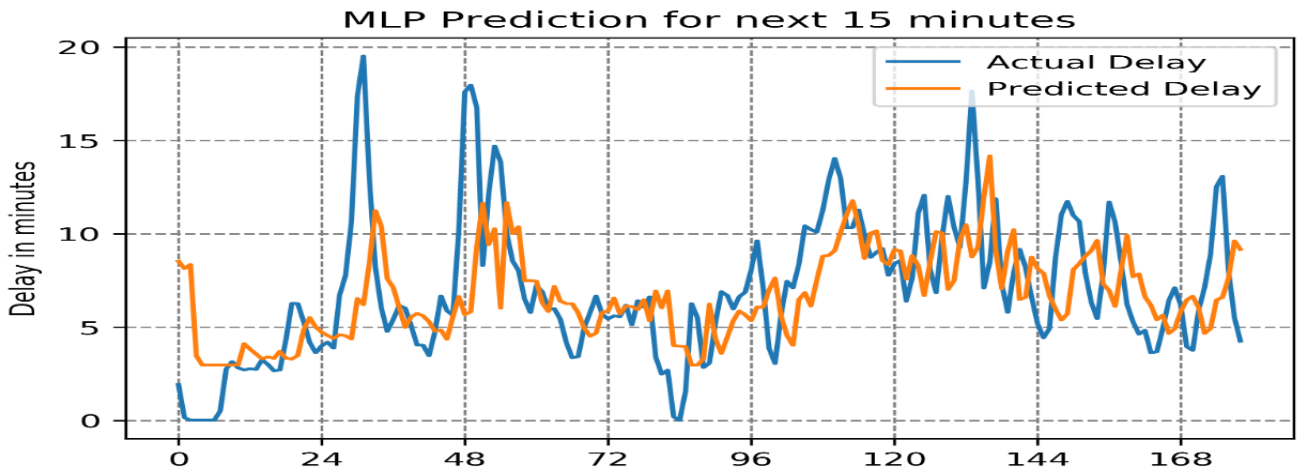


Figure C. 1: Comparing the actual U.S. bound traffic delay with 15 minutes ahead prediction of delay by MLP model at Peace Bridge for a sample of 180 data points

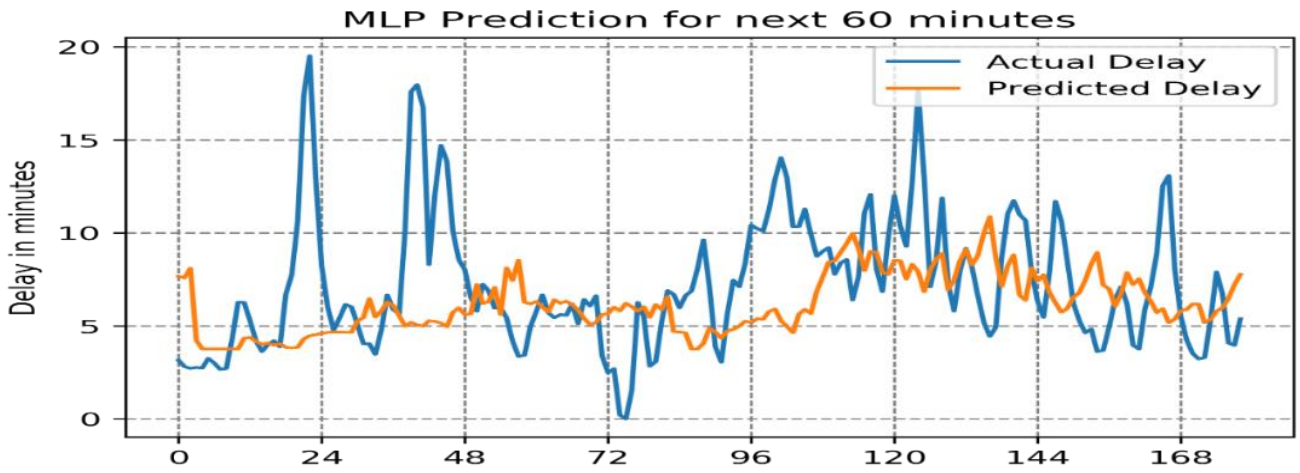


Figure C. 2: Comparing the actual U.S. bound traffic delay with 60 minutes ahead prediction of delay by MLP model at Peace Bridge for a sample of 180 data points

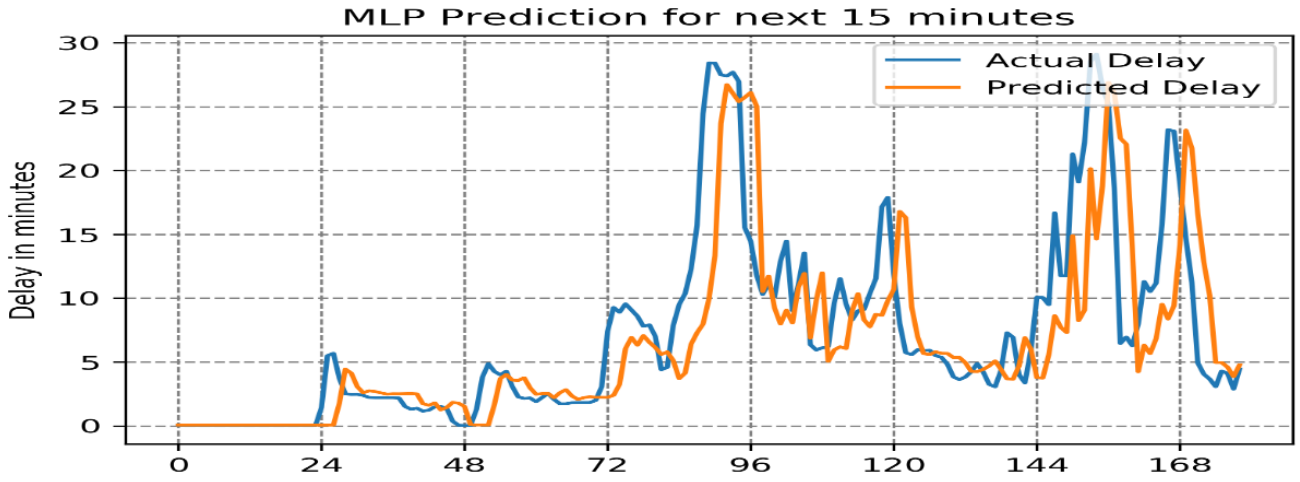


Figure C. 3: Comparing the actual U.S. bound traffic delay with 15 minutes ahead prediction of delay by MLP model at Queenston Lewiston Bridge for a sample of 180 data points

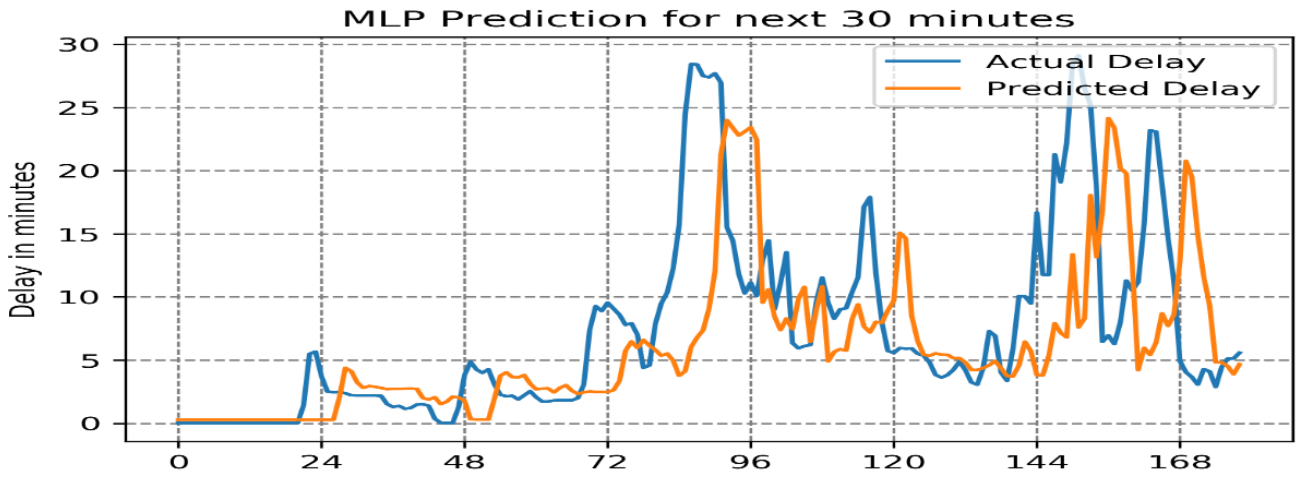


Figure C. 4: Comparing the actual U.S. bound traffic delay with 30 minutes ahead prediction of delay by MLP model at Queenston Lewiston Bridge for a sample of 180 data points

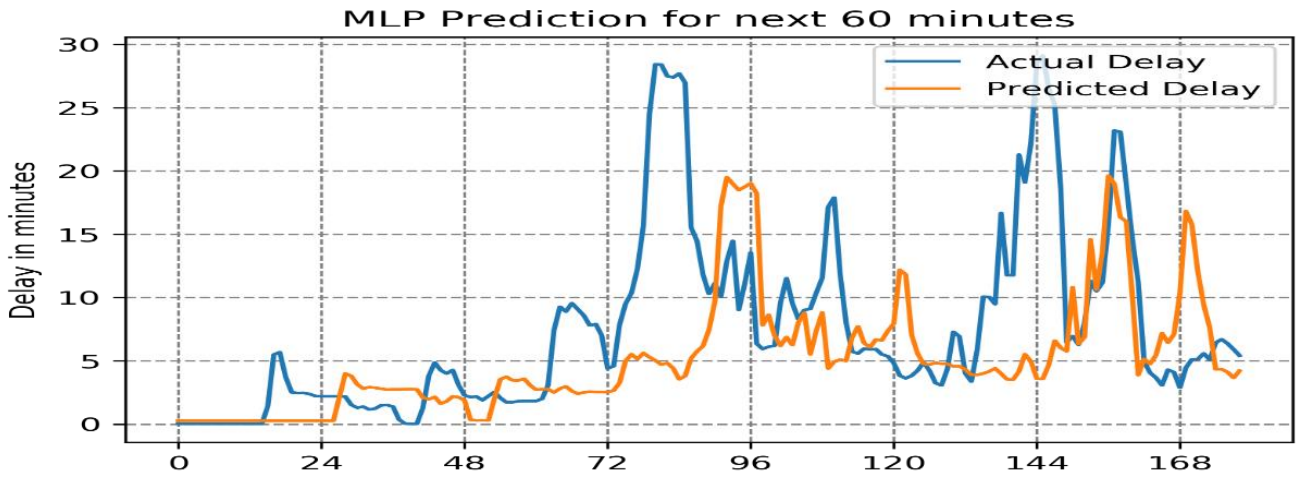


Figure C. 5: Comparing the actual U.S. bound traffic delay with 60 minutes ahead prediction of delay by MLP model at Queenston Lewiston Bridge for a sample of 180 data points

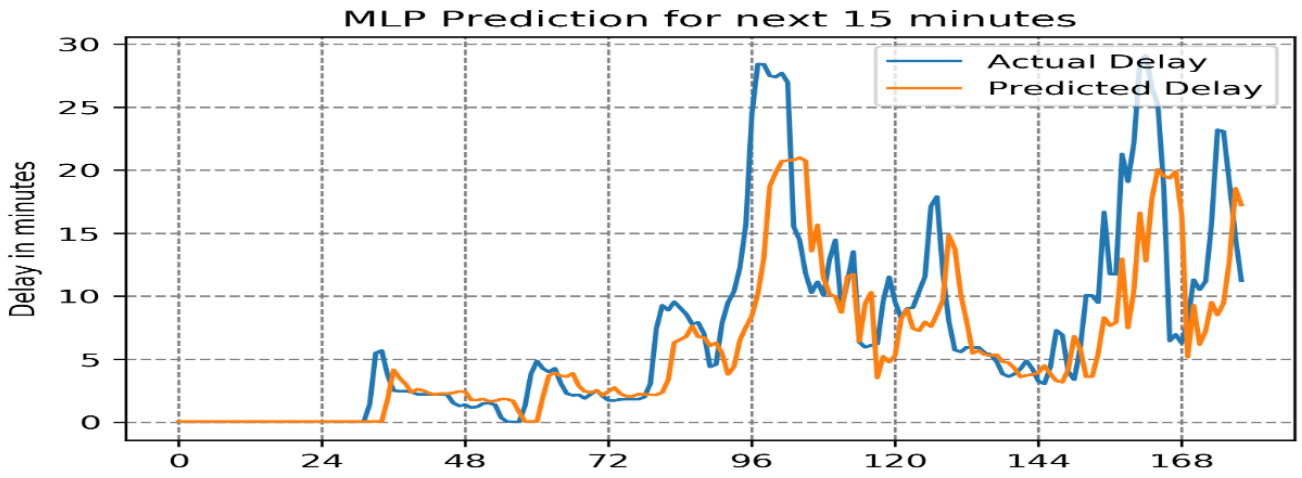


Figure C. 6: Comparing the actual U.S. bound traffic delay with 15 minutes ahead prediction of delay by MLP model at Rainbow Bridge for a sample of 180 data points

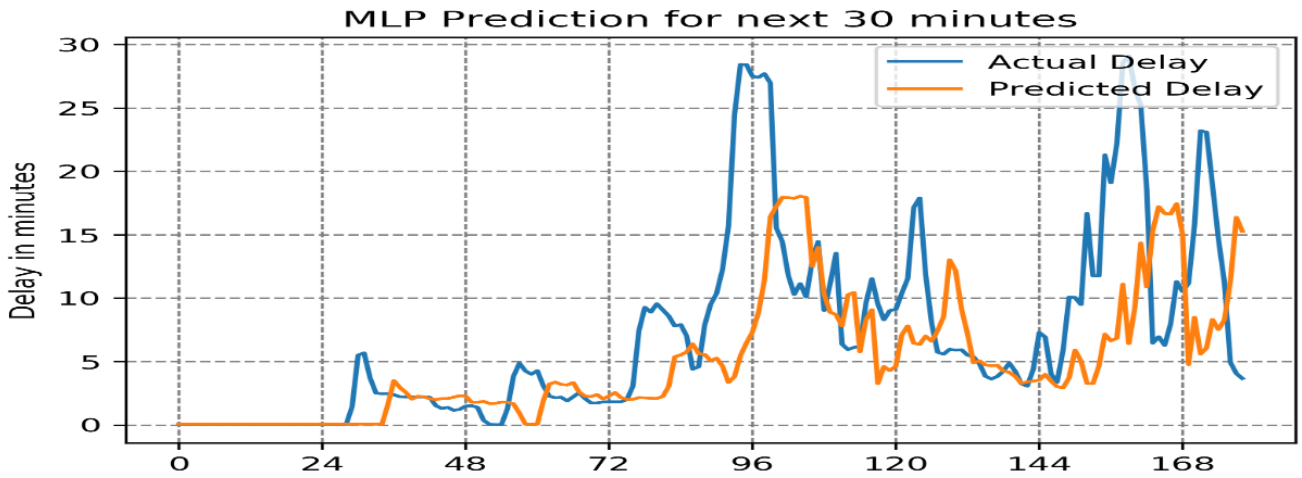


Figure C. 7: Comparing the actual U.S. bound traffic delay with 30 minutes ahead prediction of delay by MLP model at Rainbow Bridge for a sample of 180 data points

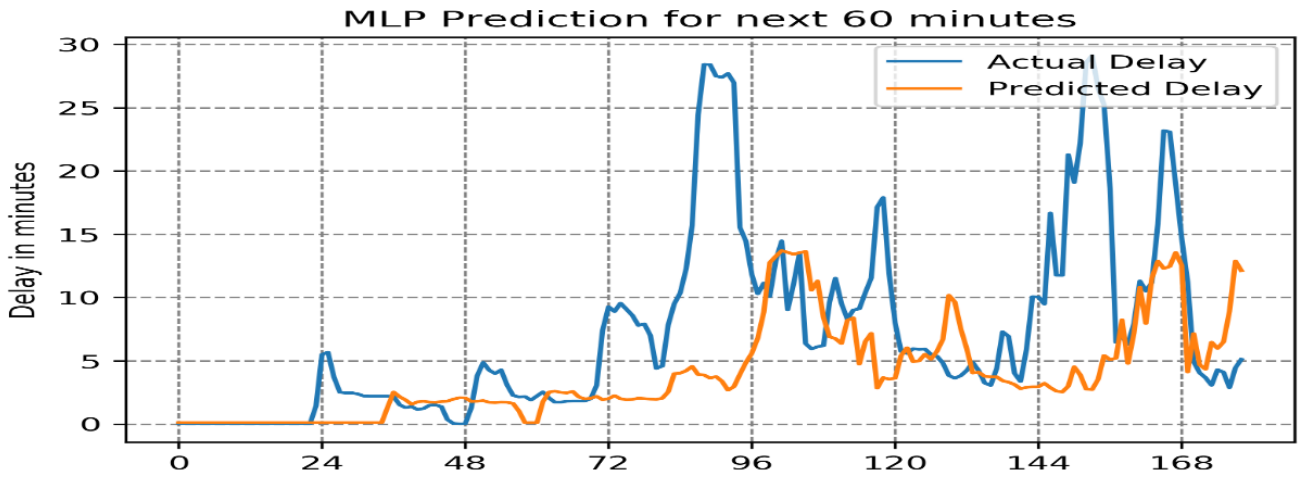


Figure C. 8: Comparing the actual U.S. bound traffic delay with 60 minutes ahead prediction of delay by MLP model at Rainbow Bridge for a sample of 180 data points

C.2 CNN model prediction results

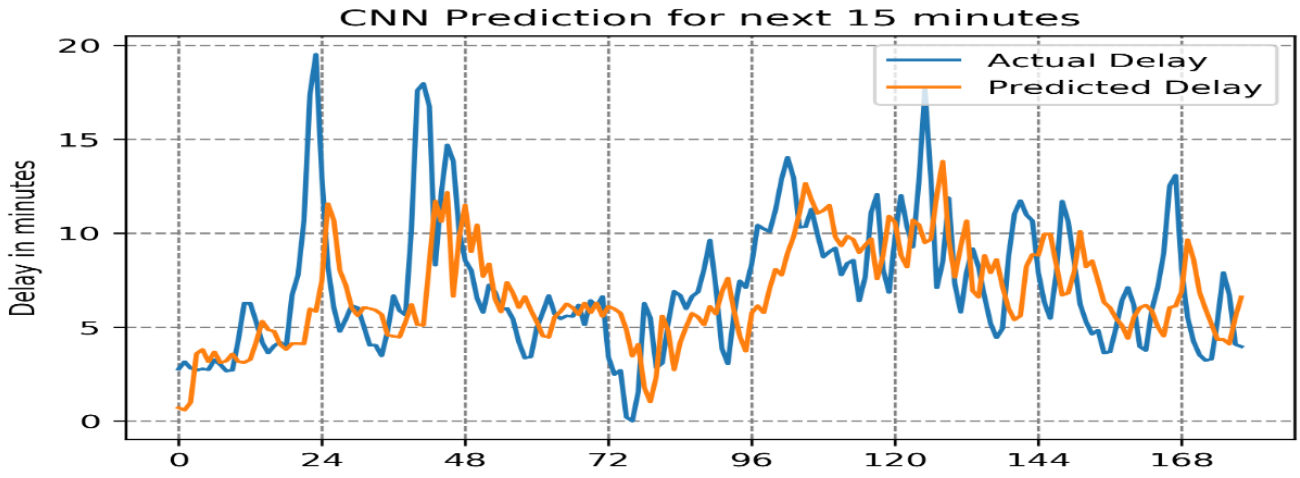


Figure C. 9: Comparing the actual U.S. bound traffic delay with 15 minutes ahead prediction of delay by CNN model at Peace Bridge for a sample of 180 data points

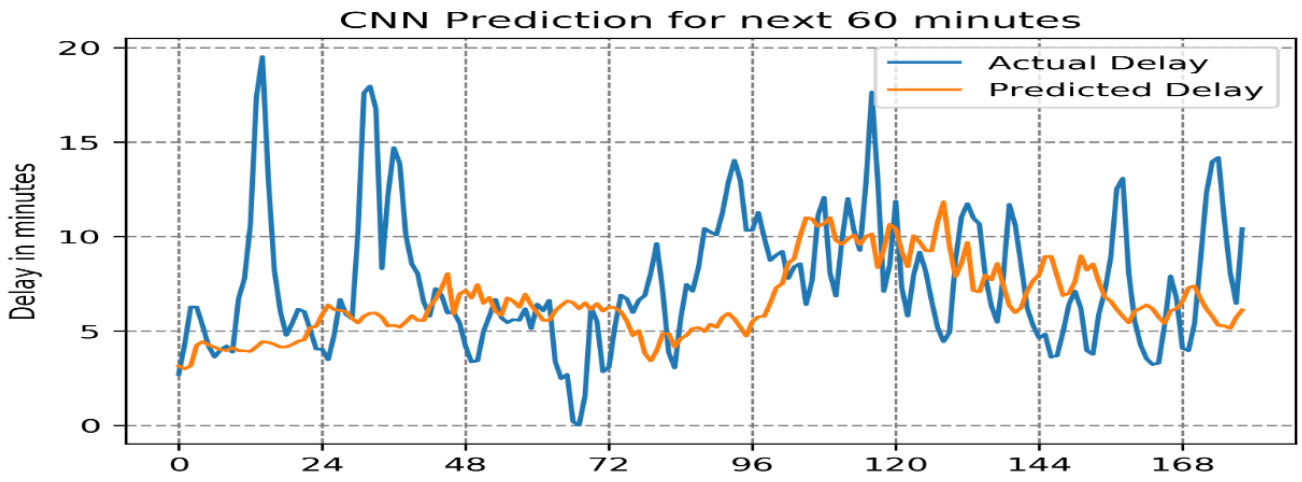


Figure C. 10: Comparing the actual U.S. bound traffic delay with 60 minutes ahead prediction of delay by CNN model at Peace Bridge for a sample of 180 data points

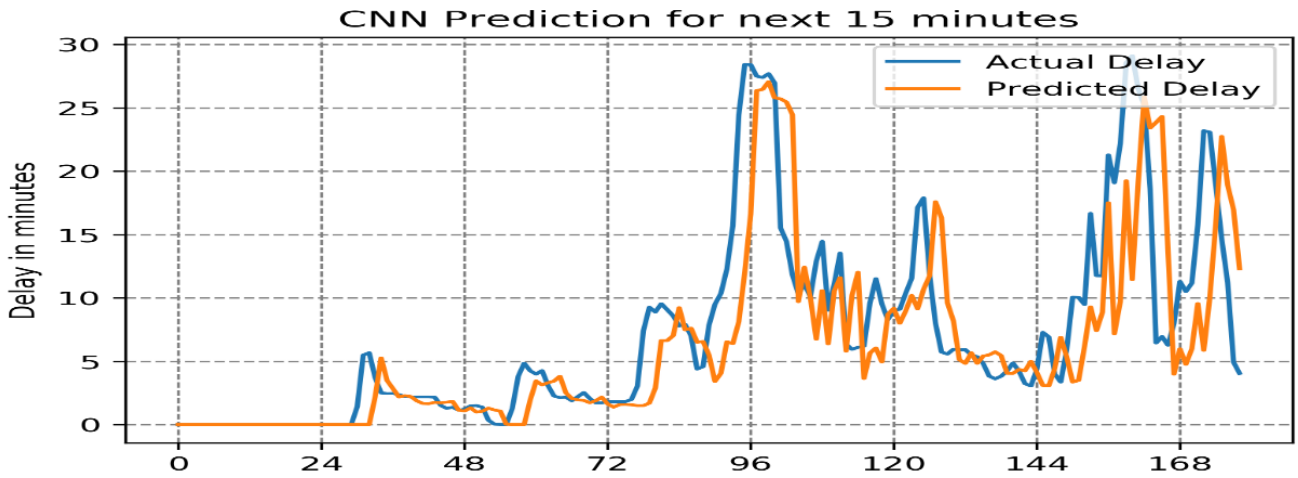


Figure C. 11: Comparing the actual U.S. bound traffic delay with 15 minutes ahead prediction of delay by CNN model at Queenston Lewiston Bridge for a sample of 180 data points

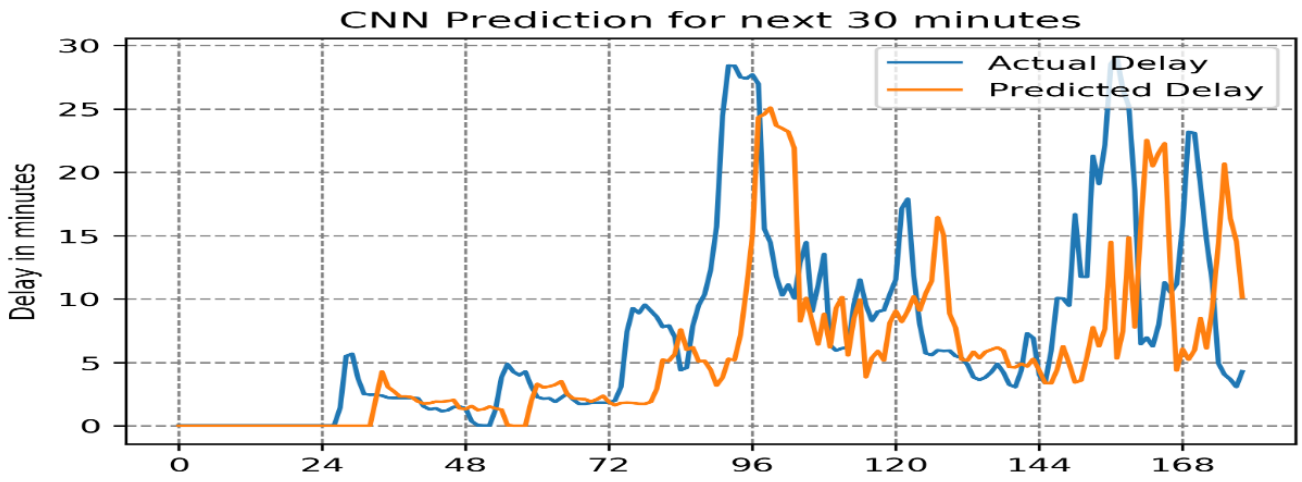


Figure C. 12: Comparing the actual U.S. bound traffic delay with 30 minutes ahead prediction of delay by CNN model at Queenston Lewiston Bridge for a sample of 180 data points

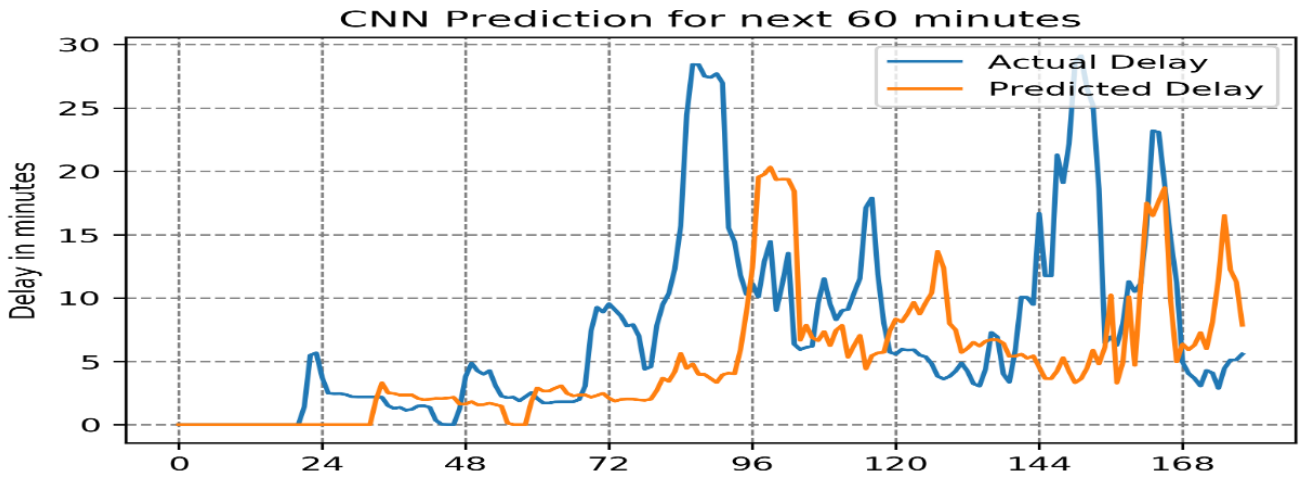


Figure C. 13: Comparing the actual U.S. bound traffic delay with 60 minutes ahead prediction of delay by CNN model at Queenston Lewiston Bridge for a sample of 180 data points

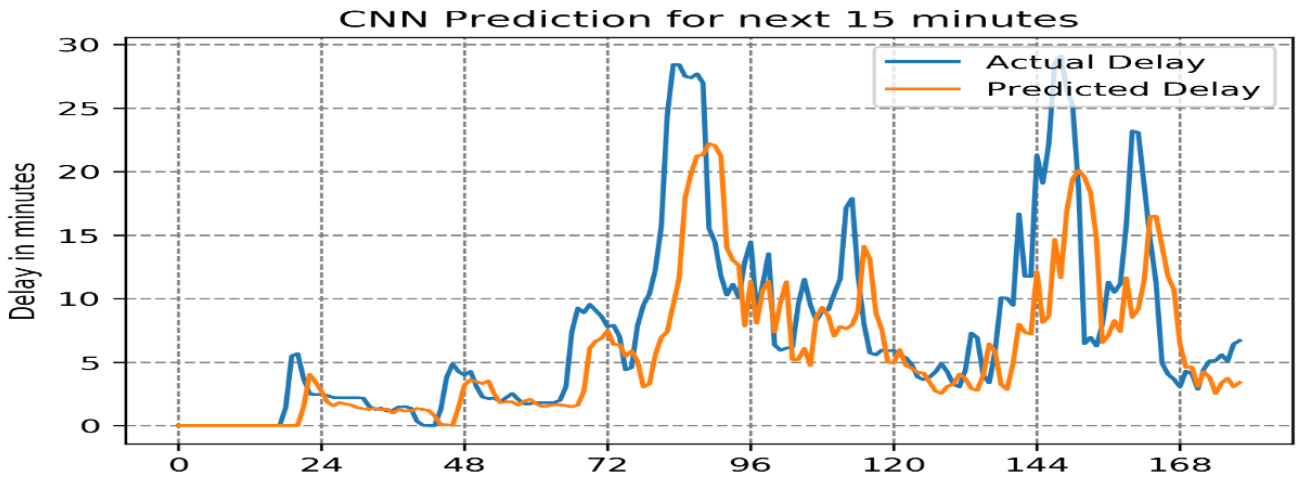


Figure C. 14: Comparing the actual U.S. bound traffic delay with 15 minutes ahead prediction of delay by CNN model at Rainbow Bridge for a sample of 180 data points

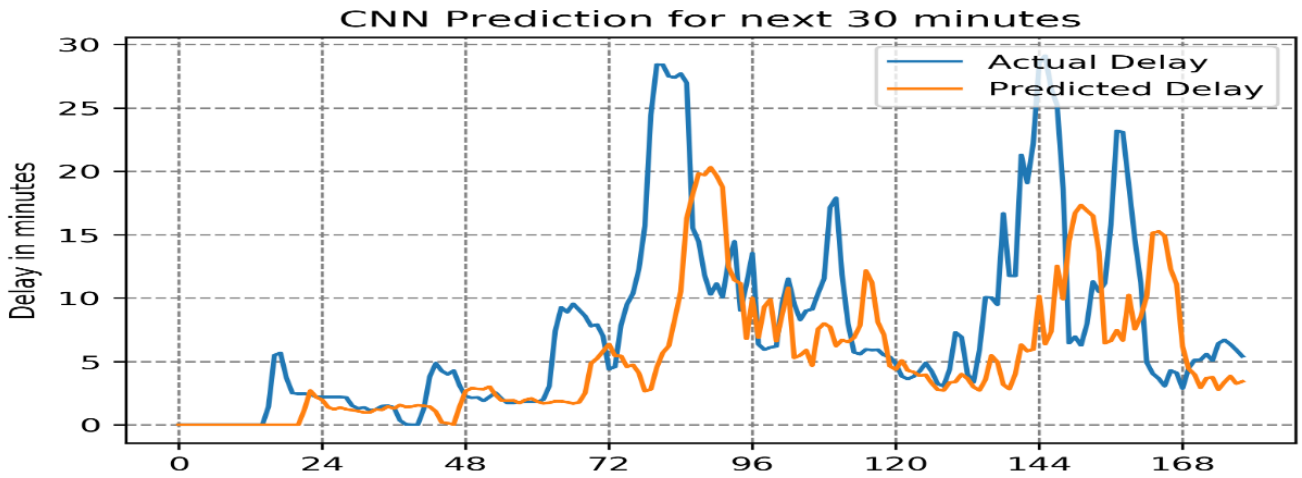


Figure C. 15: Comparing the actual U.S. bound traffic delay with 30 minutes ahead prediction of delay by CNN model at Rainbow Bridge for a sample of 180 data points

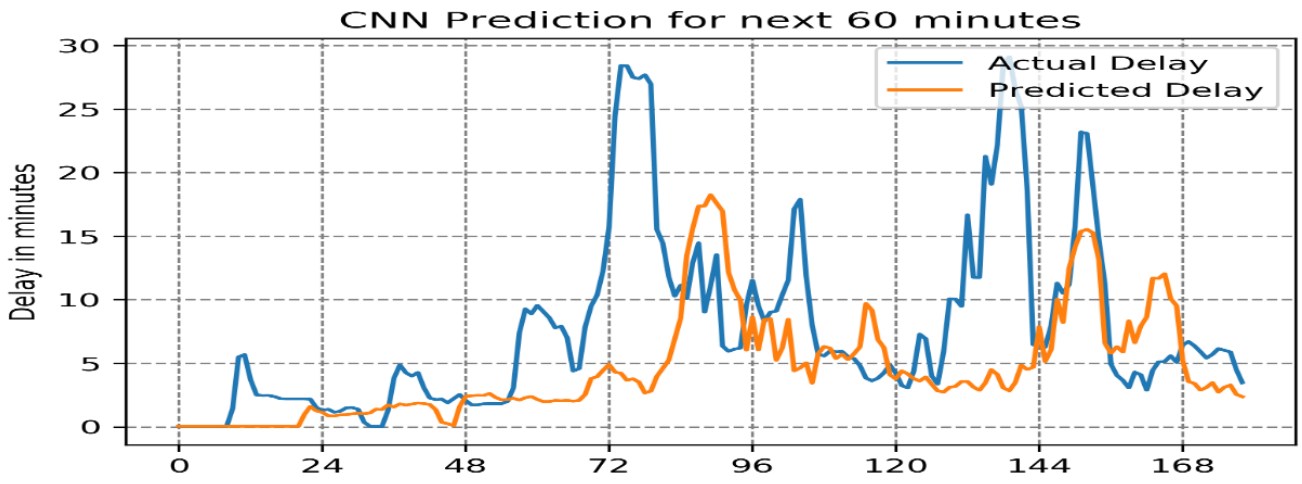


Figure C. 16: Comparing the actual U.S. bound traffic delay with 60 minutes ahead prediction of delay by CNN model at Rainbow Bridge for a sample of 180 data points

C.3 LSTM RNN model prediction results

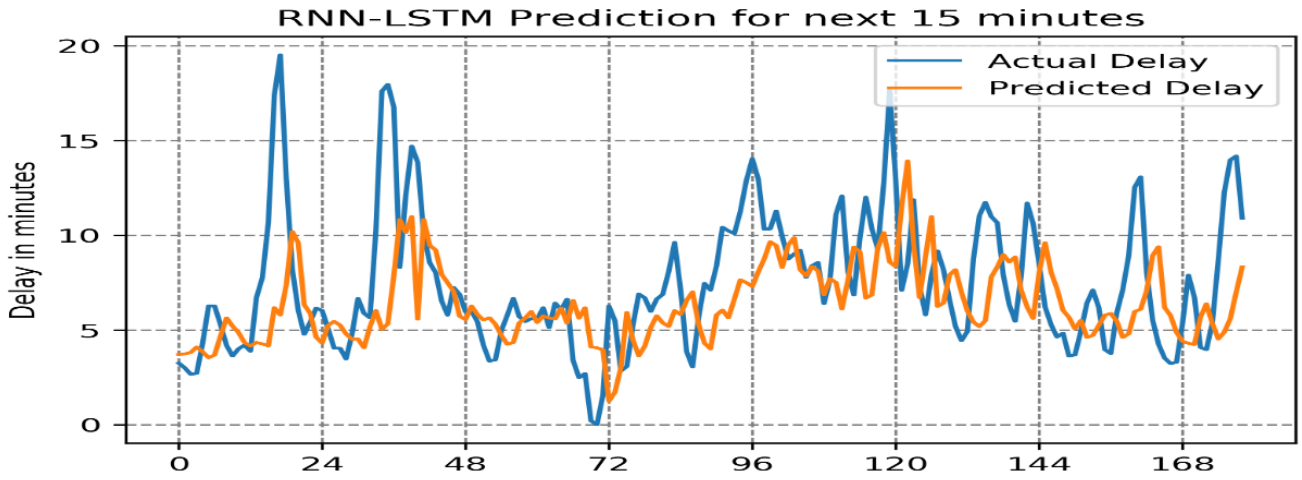


Figure C. 17: Comparing the actual U.S. bound traffic delay with 15 minutes ahead prediction of delay by LSTM RNN model at Peace Bridge for a sample of 180 data points

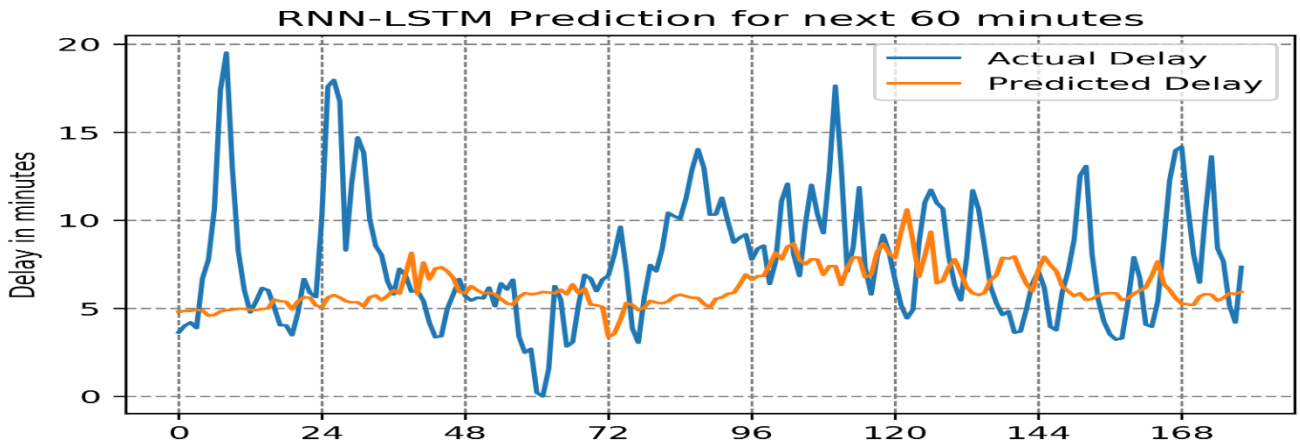


Figure C. 18: Comparing the actual U.S. bound traffic delay with 60 minutes ahead prediction of delay by LSTM RNN model at Peace Bridge for a sample of 180 data points

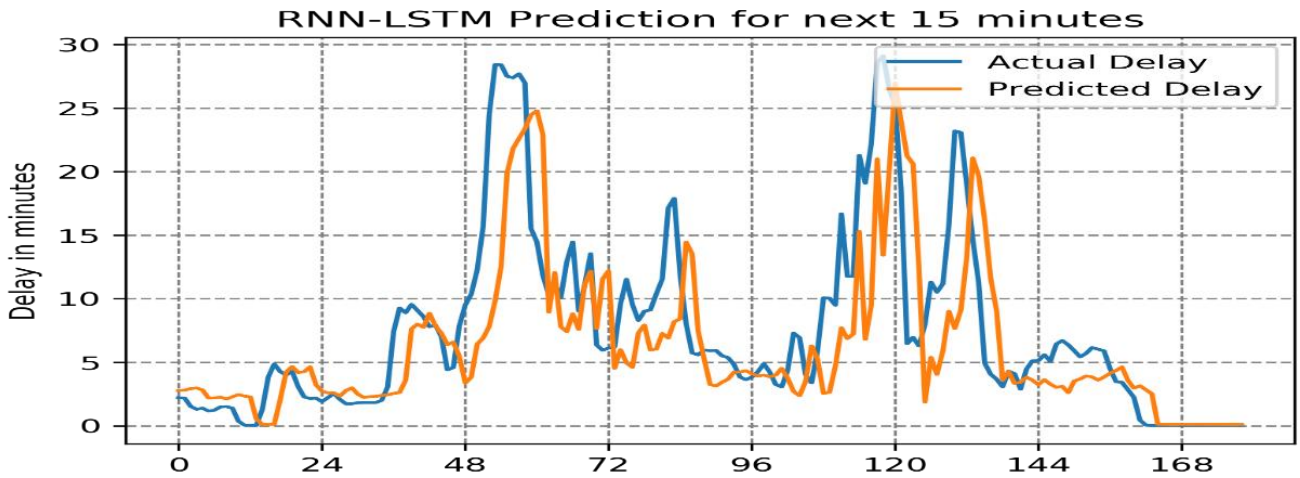


Figure C. 19: Comparing the actual U.S. bound traffic delay with 15 minutes ahead prediction of delay by LSTM RNN model at Queenston Lewiston Bridge for a sample of 180 data points

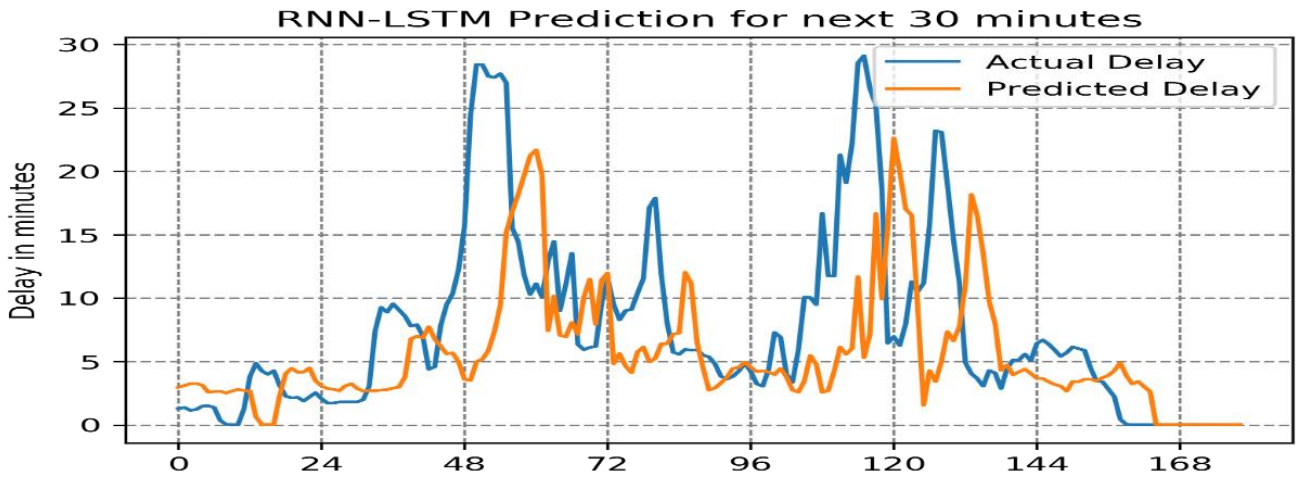


Figure C. 20: Comparing the actual U.S. bound traffic delay with 30 minutes ahead prediction of delay by LSTM RNN model at Queenston Lewiston Bridge for a sample of 180 data points

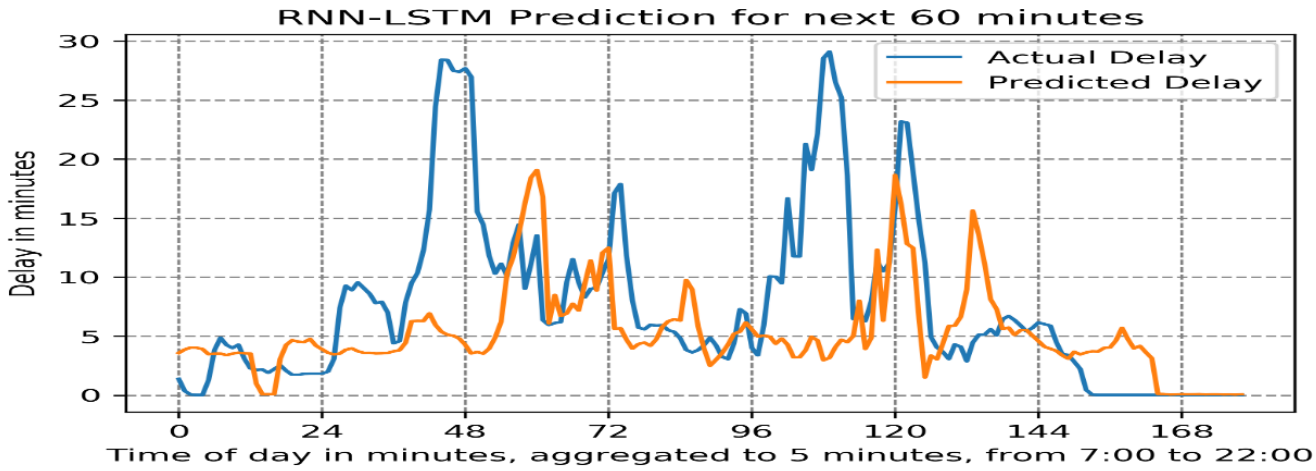


Figure C. 21: Comparing the actual U.S. bound traffic delay with 60 minutes ahead prediction of delay by LSTM RNN model at Queenston Lewiston Bridge for a sample of 180 data points

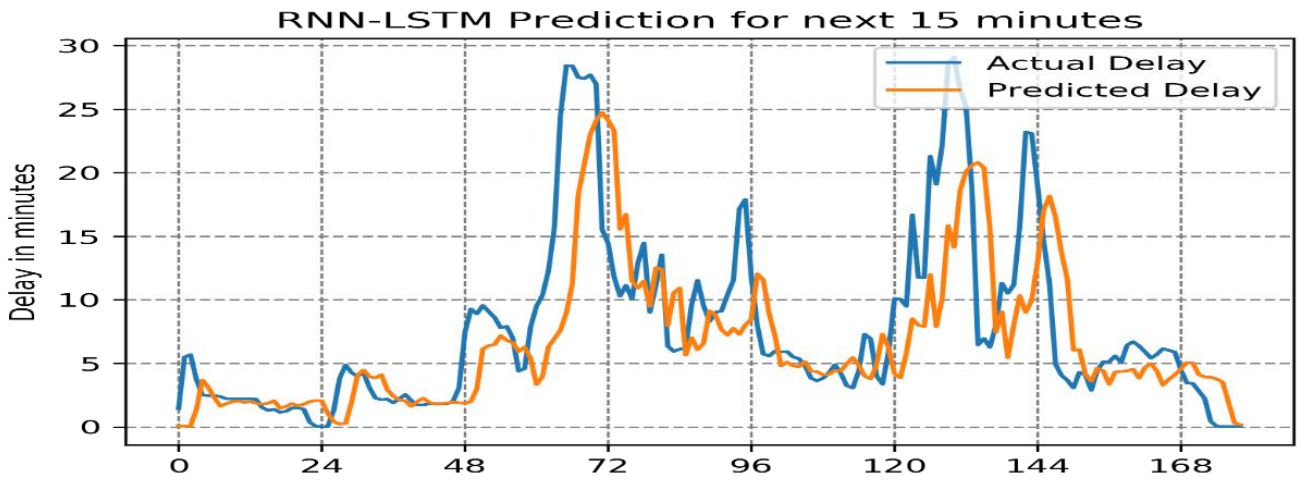


Figure C. 22: Comparing the actual U.S. bound traffic delay with 15 minutes ahead prediction of delay by LSTM RNN model at Rainbow Bridge for a sample of 180 data points

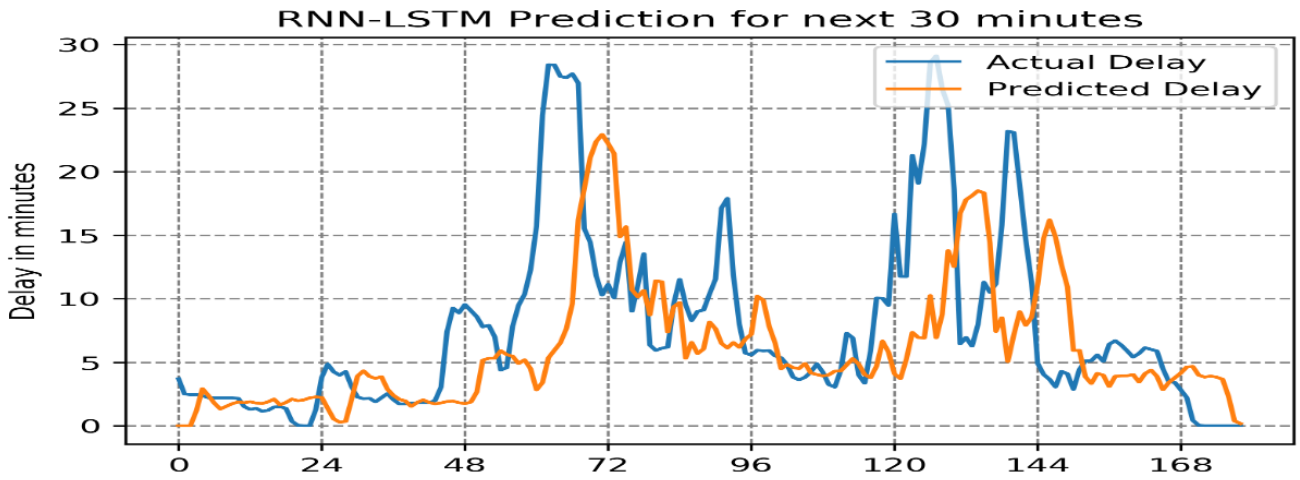


Figure C. 23: Comparing the actual U.S. bound traffic delay with 30 minutes ahead prediction of delay by LSTM RNN model at Rainbow Bridge for a sample of 180 data points

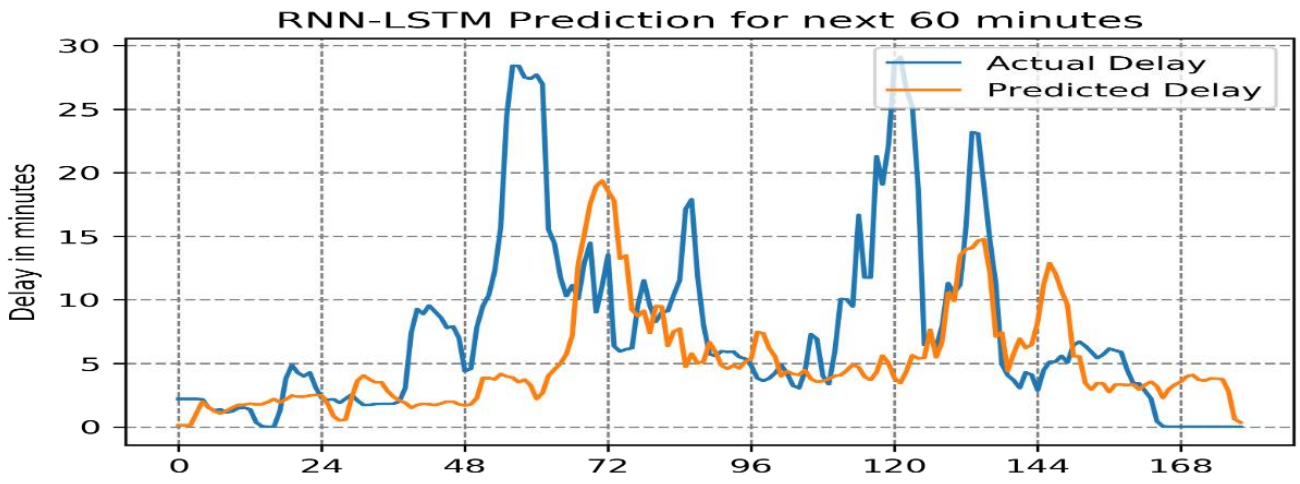


Figure C. 24: Comparing the actual U.S. bound traffic delay with 60 minutes ahead prediction of delay by LSTM RNN model at Rainbow Bridge for a sample of 180 data points

C.4 GRU RNN model prediction results

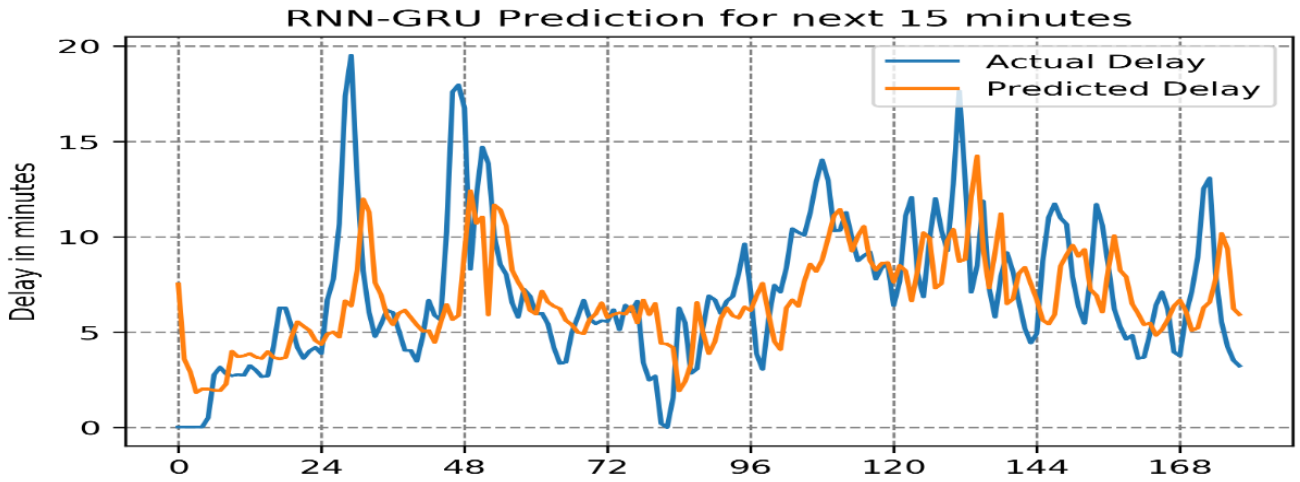


Figure C. 25: Comparing the actual U.S. bound traffic delay with 15 minutes ahead prediction of delay by GRU RNN model at Peace Bridge for a sample of 180 data points

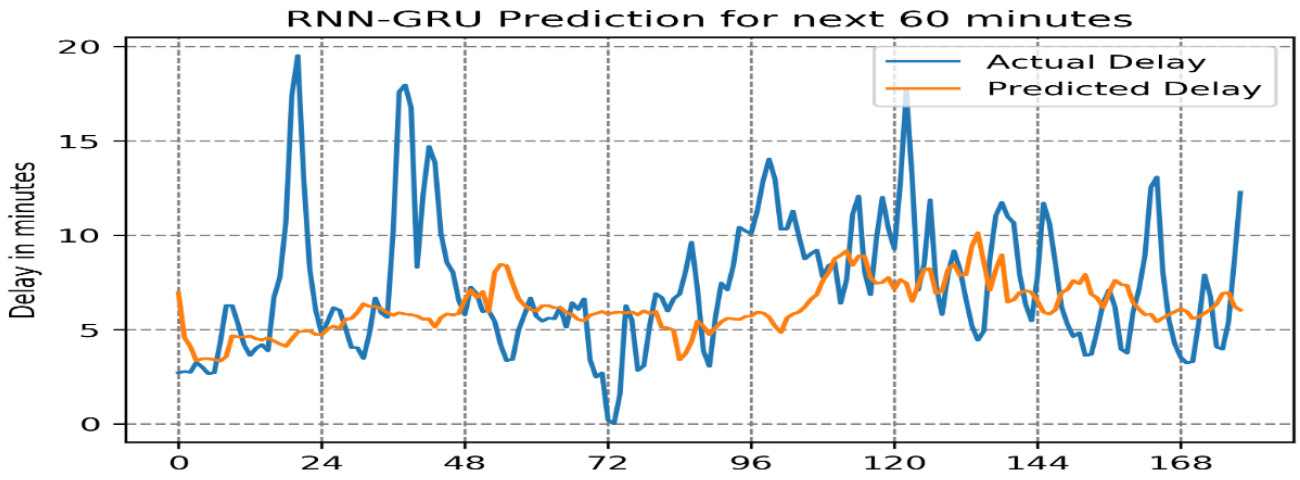


Figure C. 26: Comparing the actual U.S. bound traffic delay with 60 minutes ahead prediction of delay by GRU RNN model at Peace Bridge for a sample of 180 data points

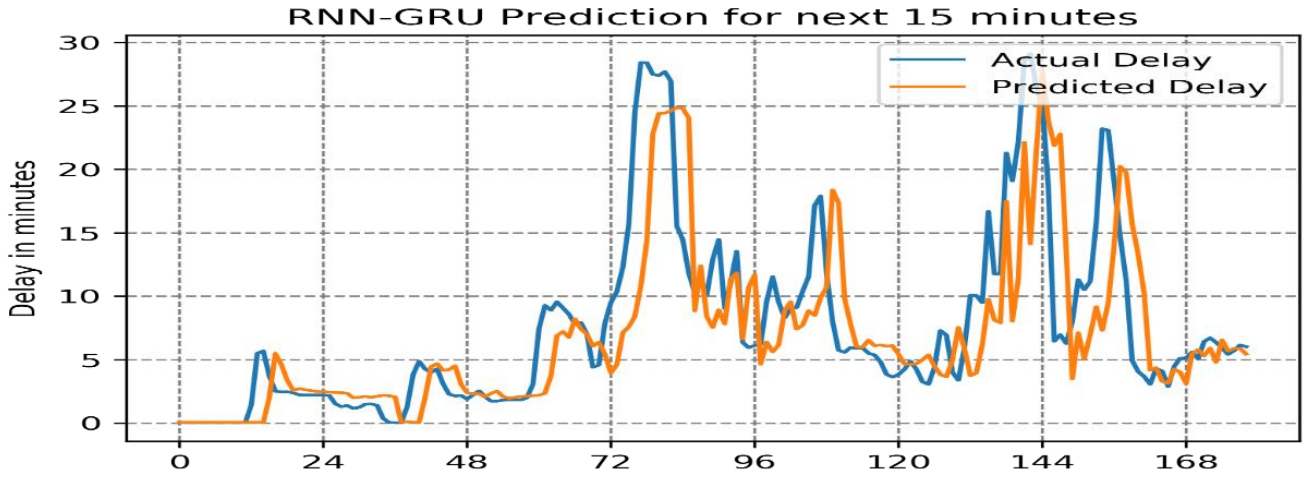


Figure C. 27: Comparing the actual U.S. bound traffic delay with 15 minutes ahead prediction of delay by GRU RNN model at Queenston Lewiston Bridge for a sample of 180 data points

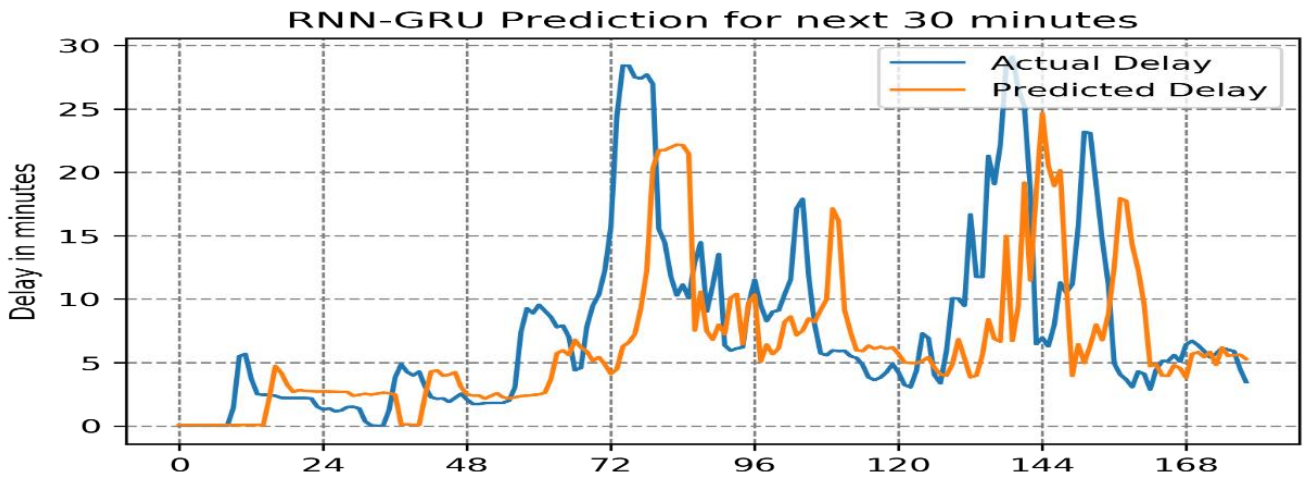


Figure C. 28: Comparing the actual U.S. bound traffic delay with 30 minutes ahead prediction of delay by GRU RNN model at Queenston Lewiston Bridge for a sample of 180 data points

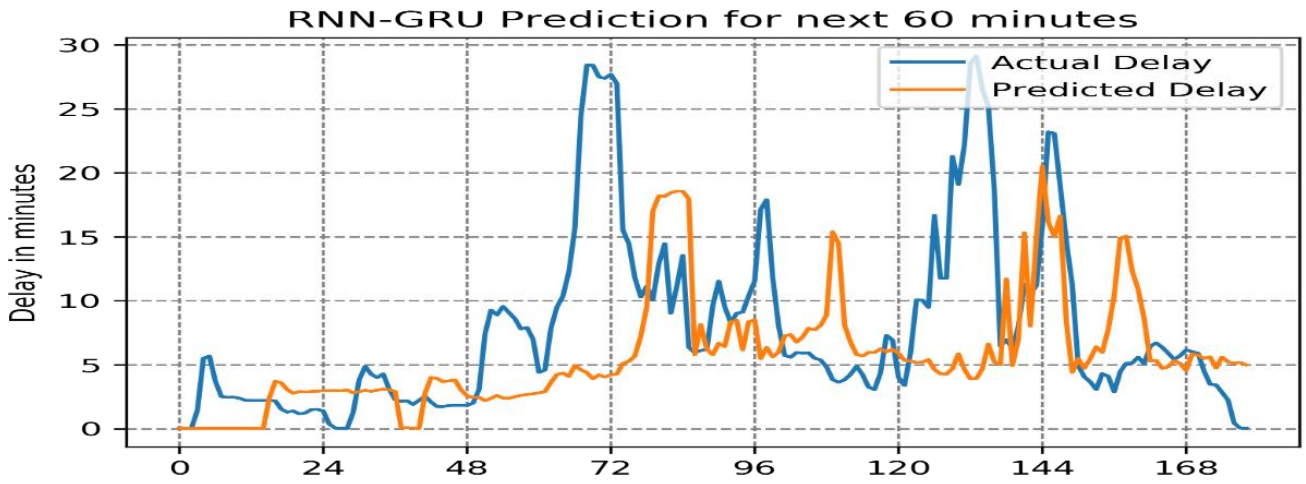


Figure C. 29: Comparing the actual U.S. bound traffic delay with 60 minutes ahead prediction of delay by GRU RNN model at Queenston Lewiston Bridge for a sample of 180 data points

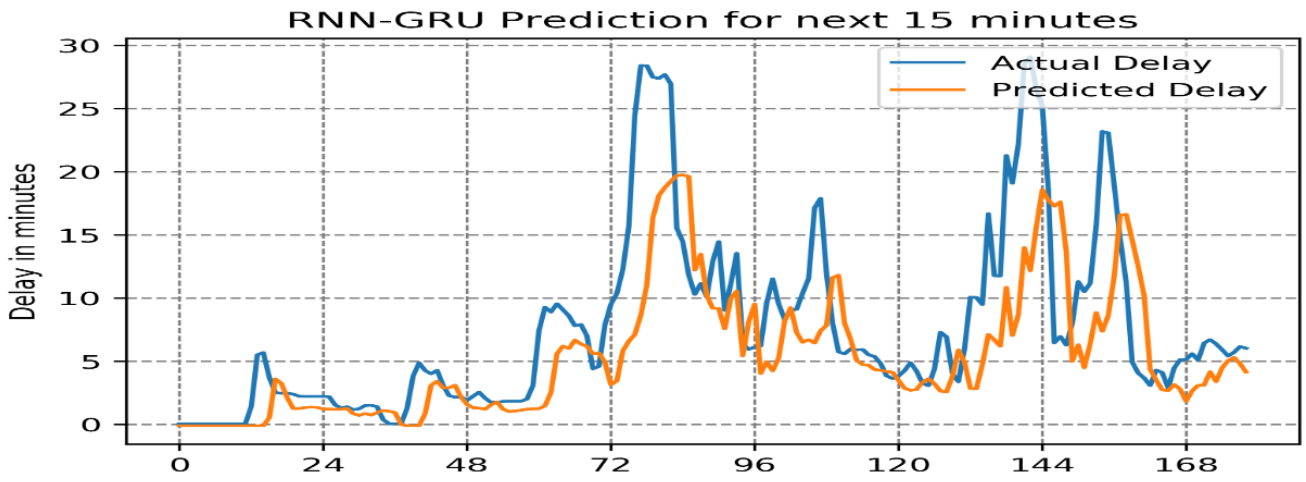


Figure C. 30: Comparing the actual U.S. bound traffic delay with 15 minutes ahead prediction of delay by GRU RNN model at Rainbow Bridge for a sample of 180 data points

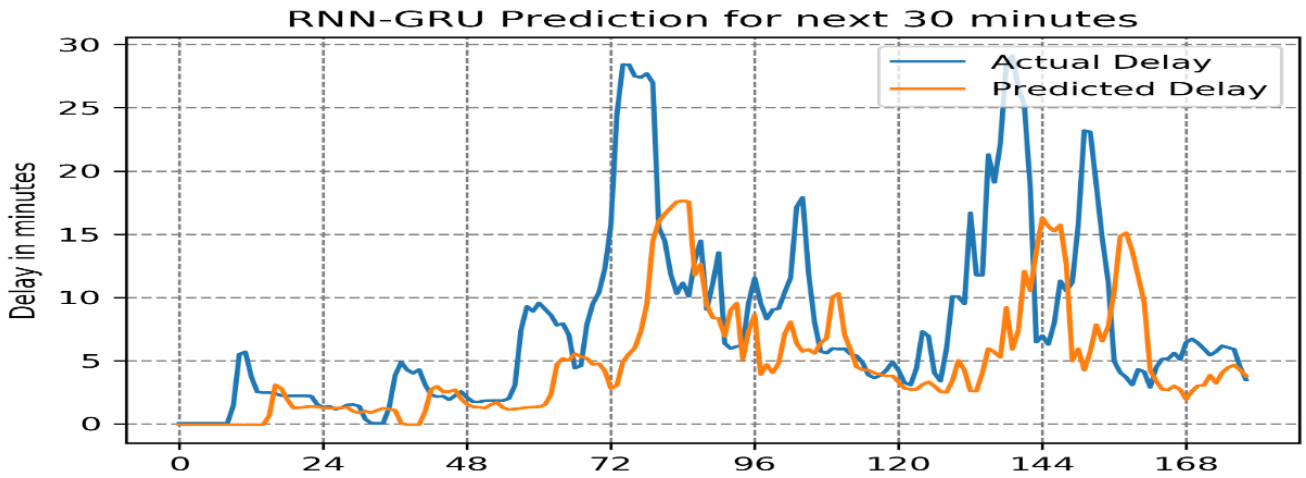


Figure C. 31: Comparing the actual U.S. bound traffic delay with 30 minutes ahead prediction of delay by GRU RNN model at Rainbow Bridge for a sample of 180 data points

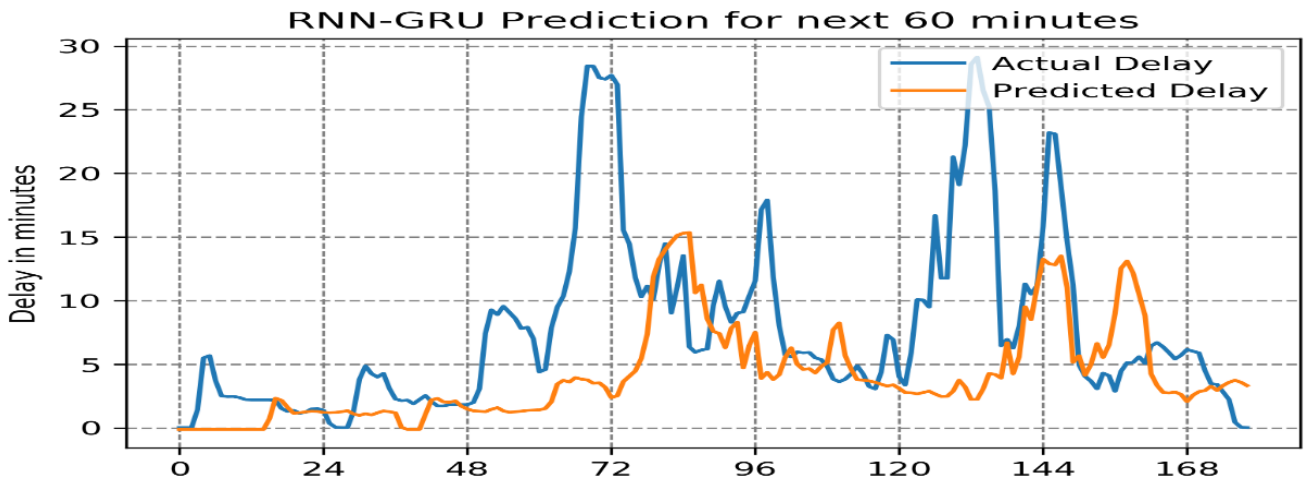


Figure C. 32: Comparing the actual U.S. bound traffic delay with 60 minutes ahead prediction of delay by GRU RNN model at Rainbow Bridge for a sample of 180 data points

REFERENCES

Aghdam, H. H., & Heravi, E. J. (2017). *Guide to convolutional neural networks: a practical application to traffic-sign detection and classification*. Retrieved from http://buffalo.summon.serialssolutions.com/2.0.0/link/0/eLvHCXMwfV1LT4QwEJ7o7sW9qKtGfKXx4G0VaAG5ug-JWT150QsBWhIjgUTA3-9Mt-yiRi8QkjZlCp1Xv_kKwN1re_JDJ5BdDyVZc5mGeSDQ6_fQckrpYVSU6SK15ZI_zkT04j9tTi6lTJIG6qRtnidF1UM5d3SJdQeR7_YxtXonD7XDANysEABUrYO-iYfR0jYM0VIFtI7tmb9JyRA5mEsHQXFsrNvyFUfP-hnd5FFSv6P-Qd3U1Og0mzf7pcq1fVrswVBR0cI-bKlyDLvdUQ3MrNwxjHq8gwdwdd--ScWaihHo3Px8ScGI3FLfNDS8PgS2mD9Po4kZPTa5HmL6vCVOP4cfwaCsSnUMTHEv8zKZ5gEGT6GfhI50gpQcH6VEIlwLLnsixZ-F3qat429yW2D9OZYFrJM91n0NrjSe300FRkyeLU7-6X4KOy7ZSp3XOINB89Gq8_XEXuhvhNfo4fULKGOn3Q

Alpaydin, E. (2016). *Machine learning: the new AI*. Retrieved from http://buffalo.summon.serialssolutions.com/2.0.0/link/0/eLvHCXMwfV3NS8MwFH9MdxGE6VScH6On3SZpknXpoQdRRxnbTl7mRV7WxIOjgmz_v3lJqkXQW0pLaEPbvI_fB4Dgd2z8658g0bq8WdkMOVMV2yiV4gRT7qIJGQX5FguxfJTlOlt1oEF7USXJA3X03lrcfrRQzi25xPhO-DYnUSNCr5WuCrK2YXModop97MwpcD2rMjipd1Fuj38MA3eB-VaQiyyVnB9Alz3AS4y_nLz9Fm6mLEAT5_LnUhQsxneQ8qvg0x0SLinfb2qhmJ9A1xF44hY6p-9BrPBuS-AmfQW_p0ZMmiXYRb-eQzJ6eH8pxnO41FnRIzlORbh8TF3CMBISvd54wV11CwnJkaLULw4yQhivUuVUuB8MNknU3H8Dgz_mu_jl3DUcuVojVhxs4pNW8_X7UoV-nLzSwjbM

Ardit. (2017, August 18). How to measure the execution time of a Python script. Retrieved March 13, 2019, from PythonHow website: https://pythonhow.com/measure-execution-time-python-code/

Bianchi, F. M., Maiorino, E., Kampffmeyer, M. C., Rizzi, A., & Jenssen, R. (2017). *Recurrent neural networks for short-term load forecasting: an overview and comparative analysis*. https://doi.org/10.1007/978-3-319-70338-1

Border delays costing billions: Ontario Chamber of Commerce laments crossing impact, says jobs and trade lost at “choke point.” (2004, July 12). *Traffic World*, *268*(28), 24. Retrieved from General OneFile.

Brownlee, J. (2016, July 20). Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras. Retrieved March 13, 2019, from Machine Learning Mastery website: https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

Brownlee, J. (2018, October 28). How to Develop Deep Learning Models for Univariate Time Series Forecasting. Retrieved March 13, 2019, from Machine Learning Mastery website:

https://machinelearningmastery.com/how-to-develop-deep-learning-models-for-univariate-time-series-forecasting/

Customizing Ticks | Python Data Science Handbook. Retrieved March 13, 2019, from https://jakevdp.github.io/PythonDataScienceHandbook/04.10-customizing-ticks.html

Dalto, M., Matuško, J., & Vašak, M. (2015). Deep neural networks for ultra-short-term wind forecasting. *2015 IEEE International Conference on Industrial Technology (ICIT)*, 1657–1663. https://doi.org/10.1109/ICIT.2015.7125335

Fu, R., Zhang, Z., & Li, L. (2016). Using LSTM and GRU neural network methods for traffic flow prediction. *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, 324–328. https://doi.org/10.1109/YAC.2016.7804912

Gabler, N., & Moens, A. (2012). *Measuring the Costs of the Canada-US Border*. Retrieved from The Fraser Institute website: http://buffalo.summon.serialssolutions.com/2.0.0/link/0/eLvHCXMwY2AwNtIz0EUrE0yMUk0TTU2SLE2NLFKAtVxiclpismFSKjArAlvs4JsUfHyMfV1MPCLN_JgY4Mu6oCNIsFISXHRDr5UCHZiIEd3ALmo8sKdsW1GUaVUAPlgXdLwROOIwha2gkWMFPmfP0BK0J93EzAJ0Ar-LdwDGRDRSveMmwDAXtnsH5EbwYqKk0rS0xJx8pJXYSEc60tknggysAYkFqUVCDEypecIMgtDFdIk5CtAiQ4RB2xc8CAmsIBWATU0F5_zikmKF_DQIB7TgLFE3NFjBCXwCqCiDoZtriLOHLmqIxCNcEG8E2oVrYWgGc4GxGANLXn5eqgSDgoVJskWSiYl5knFSskkq6B7DZGAKMDdPNTJNNk80TZZk0CLeXClSFEszcIGEQGPAhmYyDCwlRaWpsgzs0JiSA8czAD9h1WQ

Gardner, M. W., & Dorling, S. R. (1998). Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric Environment*, *32*(14–15), 2627–2636. https://doi.org/10.1016/S1352-2310(97)00447-0

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. Retrieved from http://buffalo.summon.serialssolutions.com/2.0.0/link/0/eLvHCXMwfV1LC8IwDA5uXgTBN77ZH1Dq6sJ29sEQ9eRFL7LN1JsI4v83He0QUU8ltIS0JE3SknwA0p-KycedwEFDxp4E0zRUhMHlEoooE5SizCRT-ol3u5W75Tw-4r4EFtTRNnHUaX36VIqJL21S9e8W-zEfHXA4O9cKvjlZpcqRJAKh8RyEbr4uOWAITecdS-uKHcP-zbms6-DqgoMGlOjWhJqFWfCM1bWguiS6ewbe4doGb706LOKJYXU20p8L-fwOuJzUUxc8haFSgpIZEQdVCbJhySjjYEHOMKIEe9D7yab_Z24AFR7NI8EQyrzqQaNie-P8bF4lJW0p

Harper, S., 1959, United States. White House Office, & United States. President (2009- : Obama). (2011). *United States-Canada beyond the border: a shared vision for perimeter security and economic competitiveness*. Retrieved from http://buffalo.summon.serialssolutions.com/2.0.0/link/0/eLvHCXMwY2AwNtIz0EUrE0yMgOnYMikJ2Dw3tUhKSjFONLNMSUxLArYmUo0SwVtjfHyMfV1MPCLN_JgYYNcVFoCuj0pLySkAL7F0D_DXTy_INwZWecbMDKzgXZmg1OwVBWy8JpWmpSXmIFcOboIMrKmgHQNCDEypeSIMapBWnAKkFacL3v2fqJ

AE3iiiAGxtKSSBT7sUZVBwcw1x9tCFmhgPHUcBnaJpYmoJKvXFGFiAffNUCQYF85RkYH8qMc04zcLCJ
M0sydIozdA8zdDEPMUgDXQynySDJE5jpPDISTNwQQYvQUiGgaWkqDRVFu5JObCnAatlYLY

Home - Keras Documentation. Retrieved March 13, 2019, from https://keras.io/

Khan, A. M. (2010). Prediction and Display of Delay at Road Border Crossings. *The Open Transportation Journal*, *4*(1). Retrieved from https://benthamopen.com/ABSTRACT/TOTJ-4-9

Khan, S., Rahmani, H., Shah, S. A. A., & Bennamoun, M. (Mohammed). (2018). *A guide to convolutional neural networks for computer vision* (1;1st;). https://doi.org/10.2200/S00822ED1V01Y201712COV015

Koprinska, I., Wu, D., & Wang, Z. (2018). Convolutional Neural Networks for Energy Time Series Forecasting. *2018 International Joint Conference on Neural Networks (IJCNN)*, 1–8. https://doi.org/10.1109/IJCNN.2018.8489399

Lin F. B., & Lin M. W. (2001). Modeling Traffic Delays at Northern New York Border Crossings. *Journal of Transportation Engineering*, *127*(6), 540–545. https://doi.org/10.1061/(ASCE)0733-947X(2001)127:6(540)

Lin, L., Wang, Q., & Sadek, A. W. (2014). Border crossing delay prediction using transient multi-server queueing models. *Transportation Research Part A: Policy and Practice*, *64*, 65–91. https://doi.org/10.1016/j.tra.2014.03.013

Lin, L., Wang, Q., Sadek, A. W., & Li, Y. (2014). *An Android Smartphone Application for Collecting, Sharing and Predicting the Niagara Frontier Border Crossings Waiting Time*. 5.

Ma, X., Tao, Z., Wang, Y., Yu, H., & Wang, Y. (2015). Long short-term memory neural network for traffic speed prediction using remote microwave sensor data. *Transportation Research Part C: Emerging Technologies*, *54*, 187–197. https://doi.org/10.1016/j.trc.2015.03.014

Masum, S., Liu, Y., & Chiverton, J. (2018). Multi-step Time Series Forecasting of Electric Load Using Machine Learning Models. In L. Rutkowski, R. Scherer, M. Korytkowski, W. Pedrycz, R. Tadeusiewicz, & J. M. Zurada (Eds.), *Artificial Intelligence and Soft Computing* (pp. 148–159). Springer International Publishing.

Matplotlib: Python plotting — Matplotlib 3.0.3 documentation. Retrieved March 13, 2019, from https://matplotlib.org/

Moniruzzaman, M., Maoh, H., & Anderson, W. (2016). Short-term prediction of border crossing time and traffic volume for commercial trucks: A case study for the Ambassador Bridge. *Transportation Research Part C: Emerging Technologies*, *63*, 182–194. https://doi.org/10.1016/j.trc.2015.12.004

Niagara Falls Bridge Commission. Retrieved April 3, 2019, from http://www.niagarafallsbridges.com/

NITTEC – Travel Smart. Retrieved April 3, 2019, from #

Pal, A., & Prakash, P. (2017). 5. Deep Learning for Time Series Forecasting. In *Practical Time Series Analysis*. Retrieved from https://app.knovel.com/hotlink/pdf/id:kt011LNBAP/practical-time-series/deep-learning-time-series

Polson, N. G., & Sokolov, V. O. (2017). Deep learning for short-term traffic flow prediction. *Transportation Research Part C: Emerging Technologies*, *79*, 1–17. https://doi.org/10.1016/j.trc.2017.02.024

Press Release: US Department of Transportation Unveils New Program to Fight Border Congestion, 6/2/08 | Press Releases | Federal Highway Administration. (2008, June 2). Retrieved March 27, 2019, from https://www.fhwa.dot.gov/pressroom/fhwa0812.cfm

Roelofs, T., Preisen, L., & Helgeson, C. (2016). *Performance measures and reporting for international border crossings.* (ENTERPRISE Transportation Pooled Fund Study TPF-5 (231)). Retrieved from https://rosap.ntl.bts.gov/view/dot/30813

scikit-learn: machine learning in Python — scikit-learn 0.20.3 documentation. (n.d.). Retrieved March 13, 2019, from https://scikit-learn.org/stable/

Skansi, S. (2018). *Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence*. https://doi.org/10.1007/978-3-319-73004-2

Stephenson, P. (2016). Waiting on the Canada-U.S. Border. *Mobility in History*, *7*(1). https://doi.org/10.3167/mih.2016.070118

The Canada-U.S. border: by the numbers | CBC News. (2011, December 7). Retrieved March 26, 2019, from https://www.cbc.ca/news/canada/the-canada-u-s-border-by-the-numbers-1.999207

The Python Tutorial — Python 3.7.2 documentation. Retrieved March 13, 2019, from https://docs.python.org/3/tutorial/

tpa-na Traffic Monitoring. Retrieved April 4, 2019, from http://tpa-na.com/traffic_monitoring.html

Transport Canada, & U.S. Department of Transportation – Federal Highway Administration. (2015). *Border Wait Time Technology*. Presented at the Regional Roundtable Discussions. Retrieved from http://www.thetbwg.org/downloads/7.22.15%20-%20Maritime%20Region%20-%20IrvineJulien.pdf

Zhang, Z., & Lin, L. (2017). *Abnormal Spatial-Temporal Pattern Analysis for Niagara Frontier Border Wait Times*. 11.

Zhang, Z., Lin, L., Zhu, L., & Sharma, A. (2017). Bi-National Delay Pattern Analysis for Commercial and Passenger Vehicles at Niagara Frontier Border. *ArXiv:1711.09723 [Cs]*. Retrieved from http://arxiv.org/abs/1711.09723