UNIVERSITY OF BUCHAREST

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

COMPUTER SCIENCE SPECIALIZATION

Dissertation

INTELLIGENT SYSTEMS FOR TRAFFIC OPTIMIZATION

Graduate

Iordache Florin-Nicolae

Advisor

Ichim Bogdan

Bucharest, June 2021

Abstract

Traffic optimization is a concerning problem that major urban areas are trying to solve. By using traffic flow prediction models, road managers and individual drivers would be able to optimize time spent in traffic and avoid congestions. Using data acquired from sensors placed on the roads, machine learning models can be trained for traffic flow prediction.

This project's purpose is building traffic prediction models based on data measured on roads of Los Angeles, California. The dataset is made up of measurements collected automatically using a script from the PeMS web platform developed by California Department of Transportation to make available to the public sensors data measured on the freeways of the state.

Initial work is done on visualizing extracted data. Some of the investigated perspectives are number of sensors per road, average flow per road, per day and per hour, and flow and speed measurements daily and hourly. Other perspectives are displayed to help prepare data for experiments, like visualizing how many and which sensors worked correctly during the investigated period and have measured data, instead of having imputed values.

First part of the experiments trained linear regression, feedforward neural network and tested the modeling capability of recurrent neural networks with LSTM cells with the purpose to predict average flow values of an individual freeway in a time series analysis format. The recurrent neural network had the lowest error. It was improved by having its hyperparameters optimized and then by being combined with custom handcrafted features related to temporal context – minute, hour, day of week and day of month.

The second part of the experiments analyze data at the sensor level. 40 sensors were manually selected. Initially, an RNN was trained and evaluated on data from a single sensor. Then, the sensors have been organized in a graph. A Temporal Graph Convolutional Model was built so that the spatial dependencies between sensors can be learned using graph modelling and temporal dependencies learned using recurrent models. After optimizing its hyperparameters, the performance on the test set was compared with those of the RNN trained on a single sensor's data.

Table of contents

I Introduction

This paper describes a research project that combines the domains of traffic flow theory, data science and machine learning.

Traffic flow theory is a scientific domain which uses mathematical models to describe real life scenarios from traffic and uses them to optimize it in order to avoid congestions. It does that based on data measurements taken from roads which are aggregated into large databases. Data science and machine learning are two related fields which can handle these kinds of large databases and get insights from data they contain and make use of them by creating models able to predict future data.

The purpose of the paper is exploring different ways of using machine learning models, especially most recent ones that use deep learning techniques, to predict flow of traffic as accurate as possible. The forecasts can be used for traffic optimization either by road managers or by traffic participants to optimize time spent stuck in a congestion.

The main contributions done while building this project were building an automated script that gathered traffic datasets from an online website, extracting, processing, and analyzing downloaded data and finally building and evaluating machine learning models that can forecast traffic flow.

The first chapter contains a short introduction about the project context and purpose. Second chapter describes the motivation behind the project and why traffic optimization is an important task. Third chapter describes theoretical aspects that need to be considered, starting from traffic flow theory to machine learning aspects. Chapter 4 presents the related work done regarding the traffic flow prediction task. Chapter 5 lists the technologies employed while developing the project and how they were used. Chapter 6 describes the methods for traffic flow prediction which will be implemented. Chapter 7 is about the experiments done on the dataset and examining results. Chapter 8 is the last one and is about conclusions drawn from the experiments.

Because traffic is a widely known issue that has been around for as long as the cars were accessible to the large public, traffic optimization through traffic flow prediction has been tackled before. Initial experiments have been done using statistical models like autoregressive integrated moving average (ARIMA), then basic machine learning algorithms like linear regression and multi-layer perceptron have been employed, while more recent research uses deep learning approaches like Recurrent Neural Networks or Temporal Graph Convolutional Neural Networks. This project will build models based on the most recent deep learning architectures.

II Motivation

Latest developments of human societies led to a switch in the areas where most of the people live, from rural to urban. Due to the limited space which cities have available, they become more and more crowded. In addition to that, people have been constantly improving their standards of living and, with the increased accessibility to personal cars, they have been also switching their means of transportation to personal cars. This combination led to increasing traffic in all urban areas, with lots of congestions in the major ones.

Congestion numbers have been steadily increasing as the years have passed by. A study conducted in the United States regarding urban traffic has the following conclusions [1]: while in 1982, an auto commuter was wasting 20 hours and 5 gallons of fuel each year, which meant 610 USD lost, in 2017 someone would waste 54 hours and 21 gallons of fuel, making up 1,080 USD. At a national level, the total travel delay in 1982 was 1.8 billion hours, which were worth losses of approximatively 15 billion USD. After 35 years, in 2017, these numbers have increased, reaching 8.8 billion hours' worth of time lost in traffic, which have costed 179 billion USD.

It is predicted that in the year 2025, the numbers mentioned above will increase to 62 hours and 23 gallons of fuel wasted at individual level and 10 billion hours in time delay and 235 billion USD lost at a national level.

However, if we investigate the time lost in traffic for individual cities and we look at our country, numbers start to expand. For the year 2019, in Los Angeles, California, US, a driver that commutes every day during the rush hours would waste 167 hours being stuck in traffic [2]. At the same time, in Bucharest, Romania this wasted time increases to 227 hours [3]. Therefore, Los Angeles is placed as the 31st and Bucharest as the 14th in the cities' leaderboard of traffic congestion [4]. The first 3 cities with worst traffic were:

1. Bengaluru, India
2. Manila, Philippines
3. Bogota, Colombia

We can conclude that traffic is a problem that affects all the cities globally, and that the capital of Romania ranks among the cities with highest congestion levels.

Traffic congestions cause problems both at individual level and at a community level from multiple perspectives: [5]

- Accidents are more likely to happen when there are many cars around trying to escape traffic as fast as possible.
- Air and noise pollution, both caused by vehicles.
- Reduced emergency assistance since police cars, ambulances and firetrucks have a much harder time arriving in time when they are stuck in traffic.
- Productivity which is decreased both for people and for businesses.

All of these have an impact on people’s health, wellbeing and on their country’s economy.

One major solution to improve the urban traffic may be switching from personal vehicles to public transport, bike, electric scooters or even car sharing, but there is no certainty that one can refrain from the comfort of the personal car, therefore studies can be made in order to optimize the traffic.

III Theoretical background

III.1 Traffic flow theory

The theoretical background of the task which we are dealing with has been described in the Traffic Flow Theory [6]. It is a science field which proposes mathematical models for many of the aspects regarding traffic. For this paper, the traffic stream characteristics are the ones which we are interested in.

III.1.1 Measurement methods

Several methods have been employed for measuring these characteristics. These changed over time as technology of road networks and of cars have evolved.

Measurement at a point is one of these methods. Here, a device is placed at a fixed point and is used to collect data about vehicles passing by. In the early days this was done using hand tallies or pneumatic tubes, but nowadays the most popular methods for measuring are based on inductive loop sensors or on video cameras.

To get more insights about traffic, measurements are also done along sections of roads.

III.1.1.1 Inductive loop sensors

These sensors are made up of electrical wires inserted into the road's pavement which are connected to an electric unit located outside the road. The wires are arranged to form loops. They are fed electrical energy from the electric unit with frequencies ranging between 10kHz to 200 kHz.

This configuration forms a circuit in which the wire loop is the inductive element. When a vehicle (which is made of metal) is around the wire, the magnetic field is changed which results in eddy currents that decrease the wire loop's inductance. This decrease in inductance actuates an electronic unit which then triggers a pulse that will finally signal the presence of a vehicle.

Inductive loop sensors can be setup in a single loop configuration, in which only traffic flow is measured, and the speed is then computed, or in double loop configuration, where 2 sensors

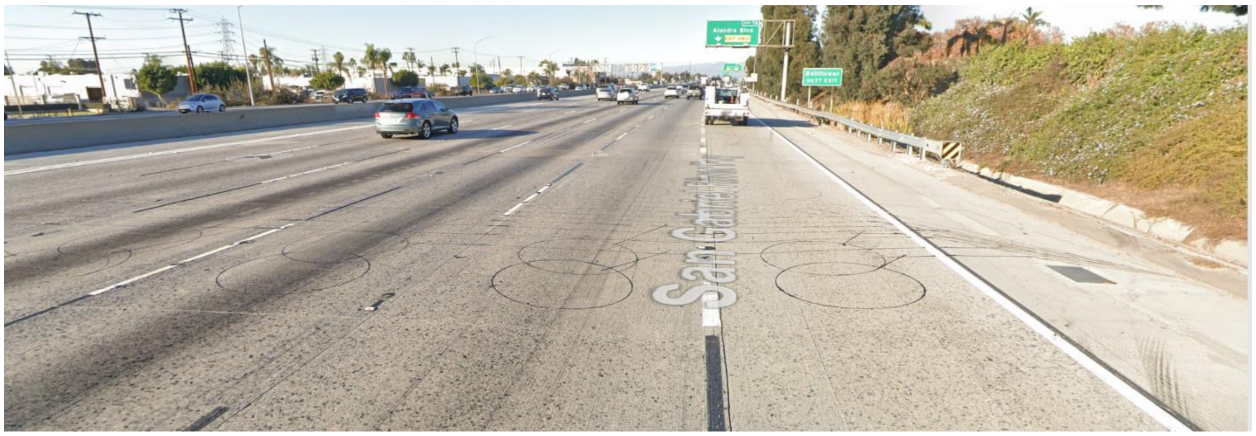


Figure III-1 – Inductive loop sensors on San Gabriel River Freeway, Lat: 33.8816818, Long: -118.1037401

(Credits: Google Maps Street View)

are placed one after the other to also measure speed of the vehicles passing by. This distance is usually about 6 meters. In Figure III-1, inductive loop sensors, the large dark circles in the road, can be observed being dug into each lane and connected to the side of the road.

III.1.1.2 Video analysis

The latest developments in computer vision and deep learning now allow traffic flow count to be made based on video feeds from cameras installed on the road. By connecting the camera feeds via the internet to servers which have high computing power, deep learning algorithms specialized in object detection with computer vision can be used for this task. [7]

However, a major drawback of this method is that it is not anonymous, because video cameras capture more information than just number of vehicles – they contain car models, license plates, and even faces of drivers. This can cause problems with data privacy laws, such as the General Data Protection Regulation (GDPR). [8]

III.1.2 Variables of interest

Modelling traffic streams is done using certain variables which are dependent either on space or time, since this kind of data is not uniform.

III.1.2.1 Flow rate

It is a quantity that represents the number of vehicles q counted in each period of time T. It can be expressed in vehicles per hour, or in shorter intervals, such as vehicles per 15 minutes or vehicles per 5 minutes.

$$q = \frac{N}{T} \tag{3.1}$$

III.1.2.2 Speed

The measurement of this variable is based both on distance traveled by vehicle and time which it took. This requires two measurement points (inductive loops) placed close to each other (6 meters), or a video recording camera, based on which average speeds can be computed, such as the space mean speed $\bar{u}_S$, which is the average time it takes to travel a given distance D.

$$\bar{u}_S = \frac{D}{\frac{1}{N}\sum_i t_i} \tag{3.2}$$

Where t_i is the time for vehicle I to cross distance D.

$$t_i = \frac{D}{u_i} \tag{3.3}$$

III.1.2.3 Occupancy

This represents how much time vehicles are covering the detector. It is computed by summing up all the time which vehicles spend on the detector during a given time interval and the dividing the sum by that time interval, to obtain the fraction of time in which the detector was covered.

$$occupancy = \frac{\sum_i \frac{(L_i + d)}{u_i}}{T} \tag{3.4}$$

Where L_i is each individual vehicle's length, d is the sensor's length, u_i is vehicle's speed and T is the time interval measured.

III.1.2.4 Density

Density is defined as number of vehicles measured along a section of the road.

$$density = \frac{N}{L} \tag{3.5}$$

III.1.2.5 Other variables

Besides the variables mentioned above, other variables which are of interest are time between vehicles, space between vehicles and concentration.

III.1.3 Modelling

Some of these variables can be combined and be used to obtain mathematical models that express their relationship. Greenshield's macroscopic stream model is one that describes relationships between several variables. [9] The modeled trends reflect real life scenarios in case of uninterrupted traffic flow.

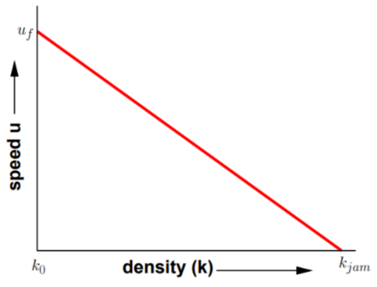


Figure III-4 – Density-speed plot

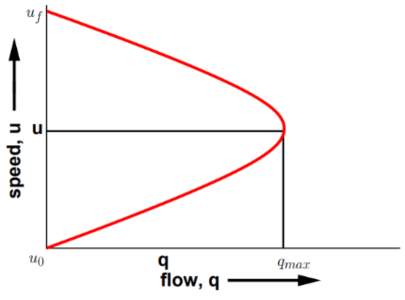


Figure III-3 – Flow-speed plot

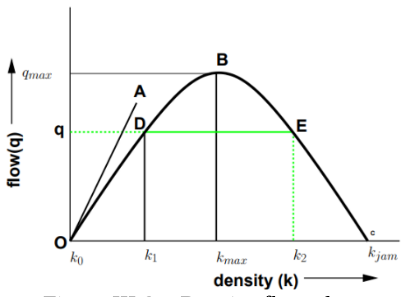


Figure III-2 – Density-flow plot

The speed-density plot shows the linear relationship between these two and that when density is 0, speed approaches free flow speed. Flow-speed plot and density-flow plot show a parabolic relationship. In the flow-speed plot we can observe that, theoretically, the maximum flow can be obtained at half of free flow speed. In the upper side of the curve is the uncongested zone, while in the lower side of the curve is the queue zone.

III.2 Traffic optimization

Traffic optimization is the set of actions which traffic managers or individual drivers can take to reduce time spent in traffic by obtaining an optimal combination of vehicles’ speed and flow.

In order to be able to optimize traffic, we need to first acquire data about it. This data should correctly communicate information like the flow and speed of traffic at certain timestamps for all kinds of situations. For example, we would need traffic information about all the hours of workdays and about weekends as well. We would also need information in all seasons of the year and in different kinds of weather conditions or when there are accidents involved.

Data needs to also be measured from the location in which we are interested. Even though there are some general patterns that apply to all major cities, the optimization needs to be as close as possible to the reality on which we are trying to improve.

When optimizing traffic, live measurements are as important as historic ones. We can combine live data with historical data to make short- or long-term predictions about how traffic will change given its current state and context and what has happened before in similar state and context.

Traffic predictions are useful for optimizing flow of vehicles when used in combination with certain actions: [5]

1. Stop lights can be changed automatically by an intelligent system which is able to consider current state and predicted state and choose which traffic path is more important to have a larger green light time in order to clear up congestion in more nodes of the city.
2. Route planning of navigation systems can benefit from predicted flows by calculating a route that considers what roads will be congested during the trip and avoid them.

Accurate traffic predictions can lead to correct decisions that decrease congestion levels.

III.3 Deep learning models

Machine learning algorithms are mathematical models able to be fit on a certain known training data set with the purpose of predicting on a new unknown test data set. This process of fitting a model to data is also named learning. There are many approaches to machine learning, the most popular are:

- Training model using examples that have labels associated – supervised learning. Examples of algorithms for this category are linear regression, support vector machines or neural networks.
- Training model using examples that do not have labels associated – unsupervised learning. Examples of algorithms are k-means or DBSCAN.

In general, machine learning algorithms are used for the tasks of either regression or classification. They are applied in different scenarios by training the models on several types of data and can be used for many applications, ranging from financial forecasting to image classification. This means that they can be used for both structured and unstructured data. The disadvantages of these models are that they require feature engineering to be done in order to be able to learn, are computationally intensive on the CPU and have the risk of overfitting, which means learning a model that is only able to handle examples from training data and not performing well on test data.

Recent research and technology improvements led the evolution of machine learning into deep learning, which employs algorithms that are able to learn tasks end-to-end, without the need of doing handcrafted features. These algorithms required 2 conditions to be met in order to work effectively:

- High computational power, which can now be obtained by doing all the intensive computations on graphics cards or on special designed hardware that is able to run the algorithms in an optimized way.
- Large number of examples in the dataset, which are now easier to obtain than before, thanks to the digitalization process through which there are lots of electronic devices like phones and surveillance cameras that are able to collect a significant number of measurements that the deep learning algorithms can use as training data.

Besides these, new ways of training the models have been discovered, which are able to avoid overfitting, like using regularization techniques such as dropout or L2.

Deep learning models perform better in tasks that are usually solved by applying classical machine learning algorithms and can also be used for new tasks, like image generation or speech synthesis.

III.3.1 Neural networks

Artificial neural networks are a very popular machine learning model that find their origin in biology, more exactly from the way biological neural networks are structured, with neurons being linked together using synapses. An artificial neural network can also be described, from a computer science point of view, using a connected directed graph, whose nodes are the neurons of the network, and each weight is an edge. The nodes compute their output based on a weight and a bias, which are both trainable parameters. It also applies an activation function to the output. Mathematically, the formula for computing the output of one single neuron with one input is described in equation 3.6, where w is the weight associated to that neuron, b is its associated bias and x is the input. The activation function is f.

$$z = f(w * x + b) \tag{3.6}$$

When plugged into a network of neurons, the output of a cell has a vectorized formula that takes into account the number of neurons that it has before and after.

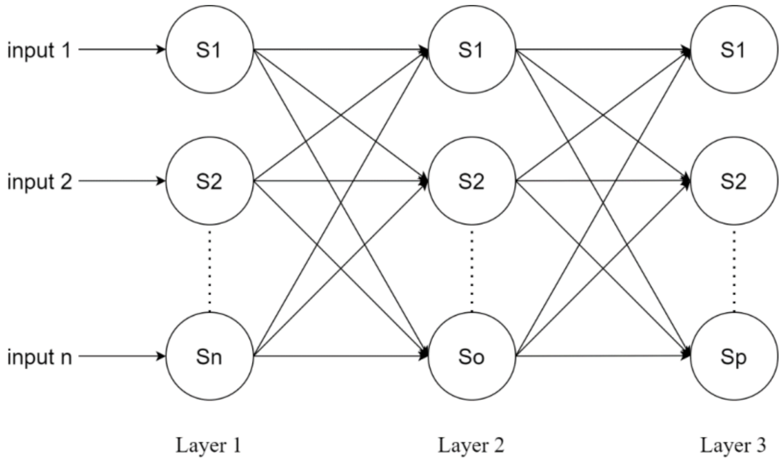


Figure III-5 – Feedforward neural network general architecture

Each cell has a weight matrix whose dimensions depend on the number of cells in the layers which are before and after. For example, a cell in layer 2 would have a weight matrix $W \in \mathbb{R}^{n*p}$ (number of cells in layer 1 * number of cells in layer 3). The bias would be a 1-dimensional vector of size p. The layers can be described as:

- First layer is a virtual layer that does not do any processing to the data, it just passes it to the network and is named input layer.
- Intermediate layers are the ones that learn an intermediate representation of the input data, and which are trainable. They are named hidden layers.
- Last layer has the role of making predictions, and it is also trainable. It is named the output layer.

III.3.1.1 Activation functions

Activation functions are mathematical functions applied to each neuron's output with the purpose of adding a nonlinearity inside the network for the hidden layers and with the purpose of bringing the output data distribution in a state which fits the task for the output layers.

Most widely used activation functions are:

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.7}$$

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{3.8}$$

$$ReLU(x) = \max(0, x) \tag{3.9}$$

$$softmax(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}} \tag{3.10}$$

When it comes to choosing the activation functions, for the hidden layers the most popular choice used to be either tanh or sigmoid, but recent papers shown that the best performances are obtained with the use of the rectified linear unit activation function, which is shortly named ReLU.

For the output layers the activation choice depends on the kind of task which is to be solved.

- For a binary classification task, output layer has size 1 and activation function is either $tanh$ or $sigmoid$ because these functions have outputs ranging between $[0,1]$ or $[-1,1]$ and by bringing a number in this range makes it easier to be classified into one of two classes.
- For a classification task, each output node is responsible for giving a confidence score for each possible class and it does this using the SoftMax activation function. This function computes each cell's output dividing current cell's value by the sum of other cells values, such that the sum of all the outputs is 1 which results in each cell representing the probability of a certain class.

III.3.1.2 Loss function

In order to be able to fit on data, the network's output is compared to the ground truth (actual label from training set). The comparation is done using a loss function, that is sometimes called cost function. The choice of this function depends on the task to be solved. Regression tasks use functions like mean squared error or mean absolute error and classification tasks use cross entropy.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_{true} - y_{pred}\right)^2 \tag{3.11}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|y_{true} - y_{pred}\right| \tag{3.12}$$

$$categorical\ crossentropy = -\frac{1}{n}\sum_{i=1}^{n}\left(y_{true}\log y_{pred} + (1 - y_{true})\log\left(1 - y_{pred}\right)\right) \tag{3.13}$$

III.3.1.3 Optimization algorithm

After computing the loss, its value is then used to compute new weights for each neural network edge using optimization algorithms. Using derivative algorithms for optimization, network's weights get updated through the process of back propagation.

Gradient descent [10] is one basic optimization algorithm. It uses the derivative of the cost function with respect to each individual weight, $\nabla\mathrm{L}(w_k)$, to update weight's value. The size of the step that the search algorithm takes is defined by α and is also known as learning rate.

$$w_{k+1} = w_k - \alpha\nabla\mathrm{L}(w_k) \tag{3.14}$$

An improved version of the gradient descent method is Adam, which stands for adaptive moment estimation. Its main feature is having a dynamic learning rate specific for each weight.

$$w_{k+1} = w_k - \frac{\alpha\widehat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon} \tag{3.15}$$

Where $\widehat{m}_t = \frac{m_t}{1-\beta_1}$ and $\hat{v}_t = \frac{v_t}{1-\beta_2}$ are the mean and the variance of the gradients computed in past iterations. So, this algorithm computes adaptive learning rates based on past gradients for each weight.

III.3.1.4 Regularization

Learning algorithms like neural networks which are able to fit data with high degrees of nonlinearity have the risk of overfitting the model to the training set and then being unable to predict on the test set. This problem is called overfitting and several methods can be used to tackle it:

- L2 normalization - Adding a term to the loss function.

$$L_{regularized}(w) = L(w) + \lambda \sum_{i=1}^{n} w_i^2 \tag{3.16}$$

- Dropout – randomly shutting down neurons of the network in order to force other neurons to do a better job at generalizing data. [11]
- Validation data and early stopping – taking a small part of the training set and making predictions on it at the end of each epoch we obtain a validation loss. By comparing this loss with previous loss values, we can observe if the model stops learning and starts overfitting. In this case the training is stopped.

III.3.2 Data normalization

Because datasets used for machine learning come from a variety of distributions, like image data, financial data, or sensor data, and have values which can be part of different intervals, they will need to be normalized [12]. For example, data from 8bit encoded images have values between 0 and 255. In other cases, there are different data intervals inside a single dataset, between features. For example, in a structured financial dataset, one feature would represent day of week, ranging from 1 to 7, while another feature would be a budget, which can have a wide variety of values.

Machine learning algorithms rely on optimization techniques which can have a harder time finding the optimal point in a dataset with different ranges of values and they may not be even capable of finding it at all. Therefore, data is normalized before training the model with it. One of the simplest ways of normalizing data is bringing all the values that have different domains into the same domain using Min-Max normalization, which takes into account, for each feature, its maximum and minimum values and the resulting data is inside the interval [0,1]. The procedure is described in Equation (3.17), in which x is a vector which contains the value of a single feature across all training examples, and $\min(x)$ and $\max(x)$ are function that compute the minimum and the maximum elements of vector x.

$$x_{normalized} = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{3.17}$$

III.3.3 Recurrent neural networks

The type of modelling employed by using basic neural networks is not ideal for tasks that require analysis of sequences of data. Recurrent neural networks aim to solve this problem by using

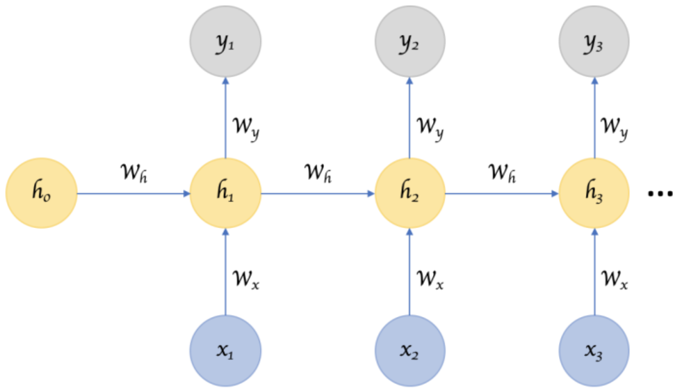


Figure III-6 – Recurrent neural network architecture. Credits: [30]

multiple networks sequentially and using the state of previous network along with current network's input for an output that depends both on current data and previous data. [13]

So, the main advantage of this type of neural networks is sharing of weights between temporally correlated data, which are called "long-term dependencies". This kind of correlation can go both ways, bidirectional recurrent neural networks can pass weights from each direction of the input sequence.

Another big advantage of this approach is the flexibility given by the fact that the input sequence can have any length.

The hidden state at the current time h_t which is passed between timesteps has the formula described in equation 3.18.

$$h_t = f(W \cdot x_t + Uh_{t-1}) \tag{3.18}$$

It takes into account:

- current timestamp's input, x_t
- a trainable weight matrix, W
- a transition matrix U that tells the importance of the present and the past inputs.
- the previous hidden state h_{t-1}

These variables are combined and are plugged into a function f that resembles the activation function from basic neural networks which can either be tanh or sigmoid.

Passing these weights repeatedly through the same activation function f causes the gradients to either explode or vanish, making the task impossible to be learned.

For avoiding this problem and for learning what kind of features should and should not be passed between timesteps, memory unit cells like gated recurrent unit, shortly named GRU, have been developed. An improved variant that of GRU that is more complex and better performing is the long short-term memory cell, shortly named LSTM.

LSTM cells are made up of gates which learn what features should be kept or forgotten and which features should be updated. The state which is passed between LSTM cells can either flow unchanged or can be updated, depending on what the gates decide. Internally, these gates are simply neural networks whose weights are trained over time, so they are learned along with the data fitting.

The gated structure of a LSTM cell in relationship with past and future timestamps is described using the visualization from Figure III-7.

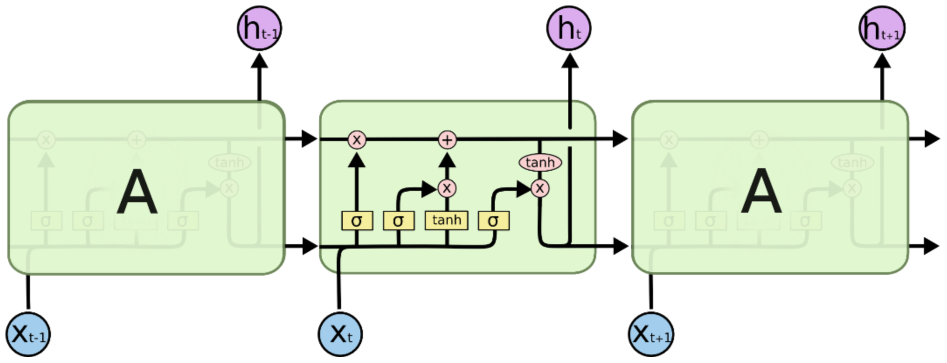


Figure III-7 – LSTM cell close-up. Credits: [31]

III.3.4 Graph convolutional networks

Deep learning can also be done in graphs. The interest in this kind of application is at a high rate since there are lots of big data modelling done using graphs, in cases like social media networks. Learning on graphs implies that each node should be able to learn its own weights and how to take into account the weights of the neighbor nodes. So, the output of each node is done by aggregating the outputs of all neighboring nodes and of the node itself. [14]

This mechanism of aggregating, named message-passing, is displayed in Figure III-8 on a graph with 4 nodes, named 1,2,3 and 4, and 5 edges. Figure is based on explanations from [15]. Each node is represented using a weight vector of size 3. The output of node 4 is shown, as an example, to be computed by aggregating the weights of all the nodes that the node is connected to and its own nodes, which are then passed through a neural network. The output size of this neural network is of a variable size n. This applies to all the nodes of the graph. So, the graph neural network can be considered as a collection of dense neural networks trained for each node.

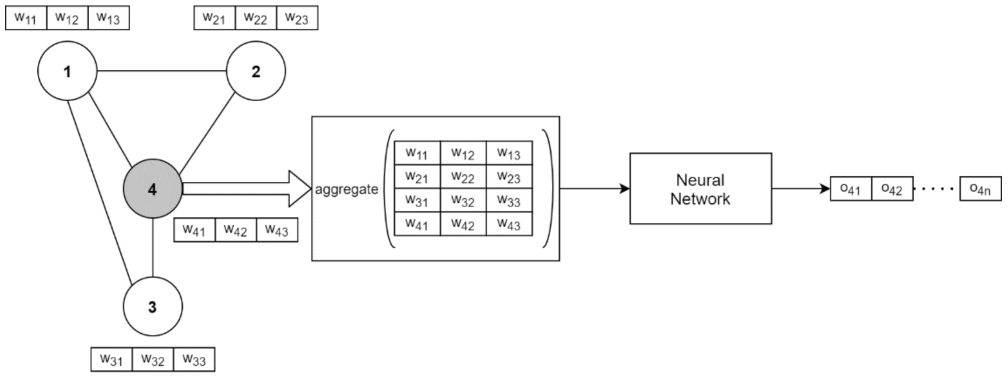
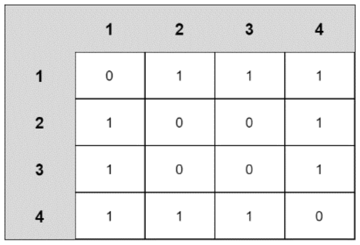


Figure III-8 - Graph neural network message passing

Graph convolutional networks can be layered together. The weights of each node from the first layer of GCN are the training input examples features, while weights of the nodes from next layers are outputs of previous layers. This means that one training input example of a GCN can be defined as a matrix X whose shape is $number\ of\ nodes * number\ of\ input\ features$. The output of the network is a matrix of size $number\ of\ nodes * number\ of\ output\ features\ per\ node$.

The aggregation mechanism is the core of these kind of architectures, and it is based on the relationships between the nodes, represented in matrix form – the adjacency matrix.

The adjacency matrix $A \in \mathbb{R}^{p \times p}$, where p is the number of sensors, is a square matrix that has value 1 for pairs of row-column that correspond to graph nodes that are linked together and 0 otherwise. It can be observed in Figure III-9.



	1	2	3	4
1	0	1	1	1
2	1	0	0	1
3	1	0	0	1
4	1	1	1	0

Figure III-9 - Adjacency matrix

Generally, the output of a layer in a graph neural network layer is computed using a function that depends on previous layer's output and the adjacency matrix. [16]

$$H_{l+1} = f(H_l, A) \tag{3.19}$$

For the first layer, $l = 0$, then H_0 is X, the input dataset.

In the context of neural network forward feed formula, the function can be written as the formula from equation (3.20), in which σ is used as an activation function and W_l is the associated matrix of weights for the layer.

$$f(H_l, A) = \sigma(A * H_l * W_l) \tag{3.20}$$

However, there are two major problems which rise from this formula.

First problem is that the adjacency matrix does not take into account a node's connection with itself (self-loop). This is an issue because a node's output should be computed based on its own weights as well, not only on neighbor. To fix this, add the identity matrix is added to the adjacency matrix to account for self-loops. By doing this, the link between a node and itself has a weight of 1, which is the maximum value that it can take, since the biggest influence on a node's output is its own state. The resulting adjacency matrix that contains ones on principal diagonal is denoted $\hat{A}$.

$$\hat{A} = A + I \tag{3.21}$$

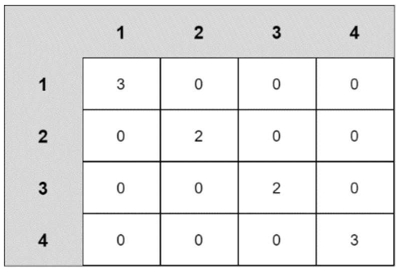
The second problem is that nodes with large number of neighbors would also have larger weights than those with less neighbors due to the adjacency matrix multiplied directly with the weight matrix.

To fix this, the new adjacency matrix is scaled with the help of the degree matrix. The degree matrix $D \in \mathbb{R}^{p \times p}$ is a squared matrix defined as equation 3.22, which says that it only has values on the principal diagonal, and the values represent the number of neighbors each node given by an index has, by summing columns of the adjacency matrix.

$$D_{ii} = \sum_j \hat{A}_{ij} \tag{3.22}$$

The degree matrix for the described graph is displayed in Figure III-10.

By multiplying the new adjacency matrix $\hat{A}$ by the inverse of the degree matrix, D^{-1}, will result in a normalized adjacency matrix. An improvement over this method is symmetric normalization, which can be expressed as $D^{-\frac{1}{2}} * \hat{A} * D^{-\frac{1}{2}}$.



	1	2	3	4
1	3	0	0	0
2	0	2	0	0
3	0	0	2	0
4	0	0	0	3

Figure III-10 - Degree matrix

The output formula of layer l will therefore have the following expression:

$$f(H_l, A) = \sigma\left(D^{-\frac{1}{2}} * \hat{A} * D^{-\frac{1}{2}} * H_l * W_l\right) \tag{3.23}$$

IV Related work

Traffic congestion is a problem present in almost every country, so the optimization of the traffic flow is a worldwide challenge that people have been trying to solve. Specific to this paper, prediction of the traffic flow has been tackled before in different ways and research is still in progress. The solutions which researchers applied have been described in several scientific articles.

Besides being a popular problem, traffic flow prediction is also a very old one, with the first article tackling the problem being published in 1979 [17].The authors proposed a statistical model for time series analysis of the traffic flow data – autoregressive integrated moving average (ARIMA) model. Several improved variants of the ARIMA model have also been tried, for example Kohonen-ARIMA (KARIMA) [18] and seasonal ARIMA (SARIMA) [19]. Another method used for modelling traffic flow was the Kalman filtering theory in 1984 [20]. These models are, however, not flexible enough to handle the non-linear way that in traffic flow values change.

To have more robust methods, machine learning models have started to be employed for solving this task, at the costs of being harder to compute and requiring more data for training. Common models which have been used are: k-nearest neighbors model [21], support vector regression [22] and artificial neural networks [23]. With the fast growth of deep learning techniques, they have been also used for this specific task, due to their flexibility and capability of learning the dynamics of traffic flow values variations and of handling big amounts of data, since the number of traffic measurements being captured has also increased substantially. The new concepts of deep learning, like recurrent neural networks with LSTM or GRU cells have been applied due to their performance in handling time series analysis [24].

These models, however, only have been dealing with temporal dependencies of the traffic flows. In reality, there are important spatial dependencies between traffic measurement sensors which need to be handled by the models. Several solutions have been proposed, that use techniques like stacked auto-encoders [25] and are able to obtain better results than previous models for short term, which means predicting flow over next 15 minutes, and long term, predicting flow over next 1 hour. Other solutions for spatial modeling make use of the combination between convolutional and recurrent neural networks [26].

Given the fact that these models are suitable for data in Euclidean space that are well structured and that traffic stations do not match this pattern since they are not arranged in a certain structure, an even more flexible kind of models would be required to better understand spatial dependencies. Modelling the data using graphs would be the best fit for the network of sensors. One of the most recent and promising types of networks being studied are the graph convolutional neural networks, which are able to learn on graphs. Recent papers have started to apply GCNs on traffic flow predictions for spatial dependencies and combined them with either Gated CNNs [27] or LSTMs [28] for obtaining temporal dependencies. These kinds of models seem to achieve better results while also being robust to perturbations.

Because at a basic level the task is a regression problem, between the employed metrics of performance used in the mentioned articles we find the mean squared error, mean absolute error and accuracy. We will be using the same metrics, but since the mentioned papers have done their training and evaluation on different datasets, current numerical results will not be able to be compared correctly.

V Technical implementation

All the work done in this project, which includes data gathering, preprocessing and analysis, and model training and evaluating is done using only the Python programming language. The choice for using this language is firstly due to the wide range of libraries it has available that help with these tasks, such as Pandas for data processing, Tensorflow and Keras for deep learning models development and other libraries which were used for this project. Second reason is the ease of use and the short time that it takes to write code and to see its results. The management of the Python version and libraries has been done using the Conda environment manager.

Data was collected from the Data Clearinghouse section of the PeMS website [29]. This section contains files with raw data from all the sensors separated by the day in which they were measured. The script uses the *requests* Python package to login to the website and to create a user session which is required since the web application is not available to users which are not logged in. By sending requests to the website which contain query parameters for district and year, it gets back a JSON response with identifiers of files that contain data for each individual day in that year. This identifier is then plugged into a template URL from each the file can be downloaded. Files collected from the website are text files, but they are formatted as CSV, so they can be easily parsed.

Using the *Pandas* package, the data loading, parsing, analysis, and preprocessing is done. With the *read_csv* method all the files have been loaded into dataframes and concatenated into one larger dataframe. Most of the data processing, like filtering, grouping, and sorting, have been done using Pandas dataframe methods, to profit from the optimizations which are done inside the package in order to do these dataframe operations faster. This is done because we are dealing with large collections of data which would take a lot of time to be manually iterated and processed with simple Python scripts.

Processed data is then visualized using either Matplotlib, Plotly or Seaborn. Simple plots like bar charts, line charts or histograms are done with Matplotlib, while more complex ones like speed-vs-flow with coloring depending on day moment and map charts are done with the help of Plotly and Seaborn is used for heatmaps.

The machine learning modelling has been done using Tensorflow and Keras. Tensorflow [30] is a computational library developed by Google for developing machine learning applications. At a basic level, it works with data using tensors, which are multidimensional data arrays, that are passed through dataflow graphs whose nodes are computations. Even if graph definition is written in Python, the computations behind it are done in C++ such that they will have a much faster execution time, given the performance superiority of C++ over Python. However, the main optimization and advantage of using Tensorflow for machine learning is its GPU computing capability. The *Tensorflow-GPU* package is able to run the computations required for training a machine learning model in parallel on CUDA cores of Nvidia Graphic Cards, or they can be optimized even further by being run on application specific integrated circuits designed for machine learning tasks named tensor processing units. In the context of this project, model training was done on a Nvidia Graphic Card, GTX 1060 6GB.

When it comes to machine learning, the library comes packaged with several tools for easily developing, training, and evaluating machine learning models, like optimization algorithms, loss functions or activation functions. For an even more faster development time of these models, another package has been built on top of Tensorflow, named Keras [31]. It contains implementations of machine learning algorithms that are very popular and can define and training models that combine them, as layers. It also comes equipped with data preprocessing functions, training algorithms, regularization techniques, activation functions and helper functions for training, like callbacks for early stopping and reducing learning rate. The latest versions of Tensorflow come already packaged with Keras. It has been used in the project for preprocessing time series data, developing and training neural network models.

The scikit-learn machine learning package was used for normalizing the data and for computing error metrics on predictions.

StellarGraph [32] is a library based on TensorFlow that aims to help applying graph machine learning algorithms. It comes equipped with several implementations of different kinds of graph machine learning algorithms, like Graph Convolutional Neural Networks. In this project, the implementation for Temporal Graph Convolutional Network, named *GCN_LSTM*, is applied on the dataset.

VI Methodology

VI.1 Time series analysis

Traffic flow prediction can be modeled using time series analysis because the input dataset contains a list of timestamped values which are chronologically ordered. To predict next values in a time series, the data must be preprocessed such that the machine learning model can take them as input and output the next one.

The length of the input cannot be infinite, so a fixed number of timesteps need to be extracted into individual training examples. This fixed number is a hyperparameter of the model and it can be set by the developer depending on how much history the model should take into consideration.

For each timestamp t of the dataset, the input values can be defined as the series $x_{t-n}, x_{t-n+1}, \dots, x_{t-2,} x_{t-1}$, where n is the fixed number of historical timestamps used for training that make up each example, i.e., the latest n timestamps.

When observing a single sensor $x \in \mathbb{R}$ and $x \in \mathbb{R}^p$ when observing p sensors.

Each training example needs to have, as output, one training example that is chronologically positioned after the ones used as input. The offset at which the output value can be found is a hyperparameter and can be set depending on how short or long term the prediction task should be. The size of the output values can also be changed such that the model to predict more timestamps into future.

So, the training example can be described using the equation 6.1.

$$x_{t-n}, x_{t-n+1}, \dots, x_{t-2,} x_{t-1} \rightarrow x_{t+o} \tag{6.1}$$

That is, n previous timestamp values of timestamp t are used to predict the value of the timestamp which is o positions after timestamp t.

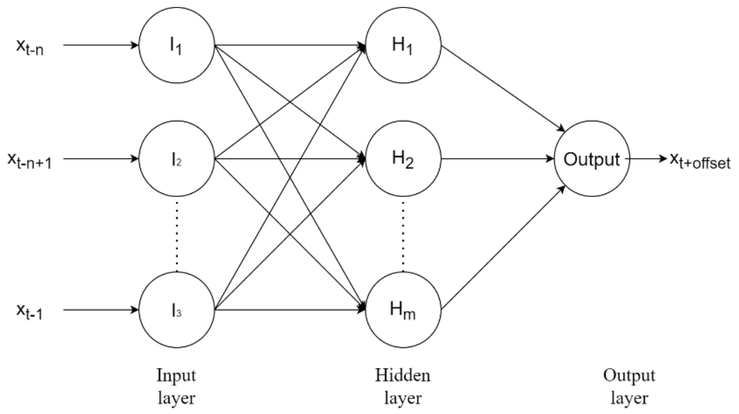


Figure VI-1 – Time series data analysis using feedforward neural network

Training a basic feedforward neural network on such a task requires an input layer having the number of nodes equal to the number of historical timestamps and an output layer of size 1. The mechanism is displayed in Figure VI-1.

When applying a LSTM recurrent neural network for the task, each timestamp would be plugged into a LSTM cell which is connected to the next cells that have input the other timestamps. In Figure VI-2. this mechanism of plugging timesteps into LSTM cells can be observed in a many-to-one recurrent neural network architecture. Each input timestamp enters in a LSTM cell which decides what state should be passed down into the next cells whose inputs are the next timestamps. All the LSTM cell outputs are combined and give a final prediction, which is the flow value after *o* timestamps.

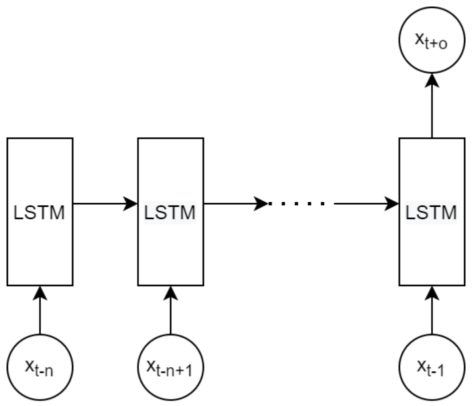


Figure VI-2 – Time series analysis using Recurrent Neural Network with LSTM cells

Predicting data is done in a similar way as training, with batches of n timestamps from the test set being passed through the network and predicting the next value. All the predicted values put together will make up the test predictions used for model evaluation. When evaluating the model and comparing the predictions with the ground truth, the first n timestamps from ground truth are ignored, since they are used for predicting the first value from the test series.

VI.2 Custom timestamp features

Keeping in mind the target task, the method of making predictions based on flow values only seems to have a flaw – the fact that it does not take into account the context of the measured value. In this case, the context is important because traffic data has certain patterns which are visible to the clear eye, and which can help the machine learning model.

The most relevant features for this task are those related to time, for example time of day, since during the night the traffic flow is low and in rush hours it has highest values. Another relevant feature is day of week, because workdays have a higher number of people driving to and from work than weekends, when people are more likely to stay at home. The features which will be extracted from timestamps and plugged into the model will be current day of month, current day of week, current hour, and current minute.

In order to achieve a model able to take into account both flow values over time and current temporal context, we append current moment's temporal features to the output of the time series

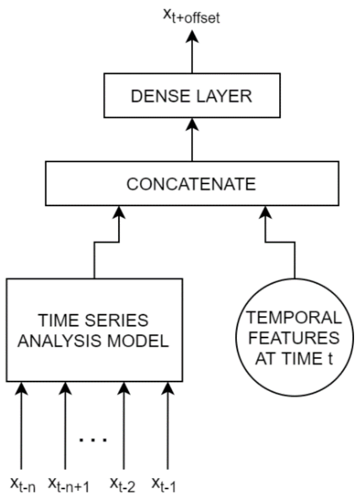

$x_{t+offset}$
DENSE LAYER
CONCATENATE
TIME SERIES ANALYSIS MODEL
TEMPORAL FEATURES AT TIME t
x_{t-n} x_{t-n+1} ... x_{t-2} x_{t-1}

Figure VI-3 – Combined time series analysis model with temporal features

model done on previous flows. The combination of these two is passed through a dense layer which will then output the prediction. This mechanism is also described in Figure VI-3.

VI.3 Temporal graph convolutional network

VI.3.1 Architecture

This architecture uses graph convolutional networks to learn relationships between sensors. It is displayed in Figure VI-4, where we have as an example a graph of 4 nodes, with each node representing a sensor measuring traffic of an individual direction of a 4-way intersection. Each node contains the value of measured flow at timestep t. The flow values of all sensors along with the relationships between sensors, represented by the adjacency matrix, are passed through a GCN. This model can be adapted such that it can be used for predicting traffic flow on next timestamp based only on flow values across the graph at current timestamp.

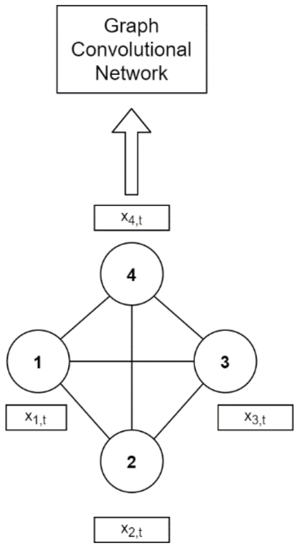


Figure VI-4 - Sensors graph at timestamp t

However, temporal data is also important for this task, so the best results would be obtained by combining traffic flow values measured at multiple previous timestamps to predict the next one. This can be achieved by using a LSTM based recurrent neural network whose inputs are the outputs of GCNs that process graph data at different timesteps. In Figure VI-5 the same sensors graph is used across n historical timestamps and plugged into GCNs that pass their output to an RNN, which is able to learn temporal dependencies between GCN outputs.

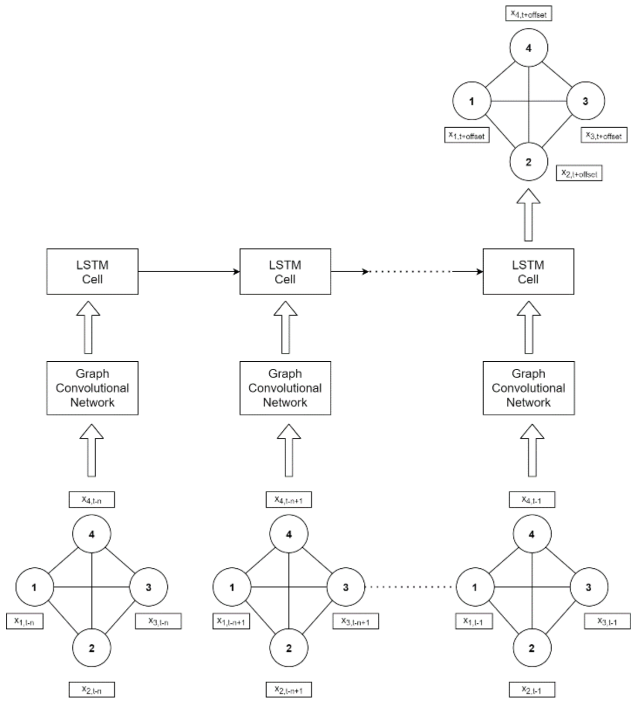


Figure VI-5 - Sensors graph at multiple timestamps entering RNN

The GCN stage of the model can be made up of multiple GCN layers stacked together to improve the performances. Same for the RNN layer, the LSTM cells can be stacked for better performances.

Model definition and training is done using StellarGraph library [32] and is based on the methodology proposed by [28].

VI.3.2 Input data

This type of model takes 2 inputs: a matrix of historical flow values of all the involved sensors, and an adjacency matrix that describes the links between the sensors.

First, the matrix of historical values is defined as $X \in \mathbb{R}^{n \times p}$, where n is the number of historical timestamps and p is the number of sensors. The structure of one training example X having n timestamps and p sensors is described in equation 6.2, where x_i^k is flow value of sensor k at timestamp i.

$$X = \begin{pmatrix} x_{t-n}^{1} & x_{t-n+1}^{1} & \cdots & x_{t-1}^{1} \\ x_{t-n}^{2} & x_{t-n+1}^{2} & \cdots & x_{t-1}^{1} \\ \vdots & \vdots & \vdots & \vdots \\ x_{t-n}^{p} & x_{t-n+1}^{p} & \cdots & x_{t-1}^{p} \end{pmatrix} \tag{6.2}$$

Then, the weights of the adjacency matrix should be introduced in the definition of the adjacency matrix such that stations that are closer together depend more on each other than stations which are far apart. The weight formula of connected nodes outputs values in the interval $[0, 1]$, and is expressed in equation 6.3.

$$w_{i,j} = \begin{cases} \exp\left(-\frac{d_{i,j}^{2}}{\sigma^{2}}\right), i \neq j \ and \ \exp\left(-\frac{d_{i,j}^{2}}{\sigma^{2}}\right) \geq \varepsilon \\ 0\, , otherwise \end{cases} \tag{6.3}$$

σ and ε are constants that can control how tightly coupled the nodes will be and can be tuned during the experiments. $d_{i,j}$ is the geographical distance between the sensors.

Distance is computed according to the Haversine formula from equation 6.4, in which $r = 6371km$ is Earth's radius, α_1, α_2 are the latitudes of the two points measured in radians, and β_1, β_2 are the longitudes of the two points measured in radians.

$$d = 2\, r\, arcsin\left(\sqrt{\sin^{2}\left(\frac{\alpha_1 - \alpha_2}{2}\right) + \cos(\alpha_1)\cos(\alpha_2)\sin^{2}\left(\frac{\beta_2 - \beta_1}{2}\right)}\right) \tag{6.4}$$

However, not all sensors are considered to be connected and some have 0 values in the adjacency matrix, before even getting to the weight formula. Such sensors are those that are immediately close to each other, but are on opposite lanes, or those that are found on different branches of the interchange, but have same directions, for example sensors that measure cars that go into the interchange from West and sensors that measure cars that go into the interchange from East are not considered connected. Even if cars have the possibility of entering an intersection from West and then from East by turning around from their direction exists, the possibility is not large enough to be taken into consideration.

The sensors which are considered connected are:

- those from same branch that have same directions – for example, all cars from West that go inside an interchange.
- those from different branches but different directions – for example cars that go inside the interchange from West are connected to those that exit the interchange to either East, North, or South, if the design of the interchange allows all these paths.

Other pairs of sensors, like those from same branch going in the same direction, or those from different branches that go inside the intersection from one and exit from another, are considered connected.

VII Experiments

VII.1 Dataset

Data exploration and model development, training, and prediction, are done on data collected from roads of the California state in the United States. They are aggregated from sensors positioned on all the freeways of the state. All the data is stored in the Caltrans Performance Measurement System (PeMS), a web platform where users can analyze and export traffic measurements. This data was automatically collected from the website using a Python script.

Sensors used for vehicle detection are inductive loop detectors buried in freeways or radars. Most of them, however, are single loop detectors. They are able to keep track of the flow of vehicles in a time period and of occupancy - average time which the detector is on. There is one loop on each lane. All the loops from a certain location and direction that are spread across all lanes form a vehicle detector station (VDS), which is a logical grouping of the detectors. Almost all the calculations which are done in the PeMS system are done at the VDS level. Many VDSs are then connected to loop detector stations, which are physical cabinets on the side of freeways that aggregate signals from sensors.

The measurements are done every 5 minutes and contain data aggregated across all lanes for flow, speed, and occupancy, the 3 main variables of interest specified in traffic theory. They will be used for modelling.

Besides these variables, data also contains:

- timestamp of measurement – in the format day/month/year hour:minute:seconds
- station ID – internal unique station identifier used in PeMS
- road ID – road number in which the station is part of (it is also the number which drivers see on signs)
- direction of travel – North, South, West, East
- type of lane - FR (Off Ramp), HV (HOV), ML (Mainline), OR (On Ramp)
- percentage of observed timestamps - because the data comes from real life sensors, they can sometimes have faults which result in data not being measured. The PeMS system

handles data imputation for times when sensors ran into faults and the percentage of observed timestamps represents how many of all the timestamps are actual measurements and not imputed values.

Sensor data are separated by multiple criteria, such as direction of travel, freeway number and district. Districts are geographical regions of the California state in which it is split into.

Even though single loop sensors are not able to measure speed, the PeMS system is able to compute it, based on flow, occupancy, and a g-factor. The g-factor is a value that represents the effective length of the vehicle. It is a combination of the average length of the vehicles in the traffic stream and the tuning of the loop detector itself. It is estimated every 5 minutes over an average week.

Measured data is aggregated every 5 minutes. Because every variable is measured per lane, the aggregation for the entire group is done as follows:

- Flow is the sum across the lanes.
- Occupancy is the average across the lanes.
- Speed value is the flow-weighted harmonic mean speed.

From a quantitative perspective, when talking about 1 single sensor, a full day of measurements, one every 5 minutes, is equal to 288 timestamps. Freeway #405, for example, has 644 associated sensors, which means that 185,472 timestamps are registered every 24 hours. When scaling up to modelling using monthly or even yearly data, the magnitude of the data count gets very large. Deep learning models, however, can get the most out of such large numbers and learn from it to make accurate predictions.

The PeMS system also offers a metadata file which contains structured descriptions of each individual sensor in a certain district which were available in a certain day. From the information provided per sensor can be mentioned: freeway number, direction, latitude, longitude, or road type.

Downloaded files are text files formatted in CSV format, which contain all sensors data for a specific day.

VII.2 Data visualization

For this paper, the focus is on sensors from District 7 (Los Angeles) in April 2019. Sensors which are found only on mainline will be taken into consideration and some preprocessing will be done to only work with data which is mostly observed and not imputed.

After downloading the data from PeMS website, the files have been parsed and loaded into Pandas dataframes. Several analysis perspectives have been employed to obtain an overview of the context and to get insights about traffic data in the zone.

In the analyzed period (April-June 2019), District 7 has a total number of 4864 installed sensors on 22 freeways. The sorted number of stations associated with each freeway is displayed in Figure VII-2. The freeway with largest number of sensors is the Interstate 405 (San Diego Freeway). It is

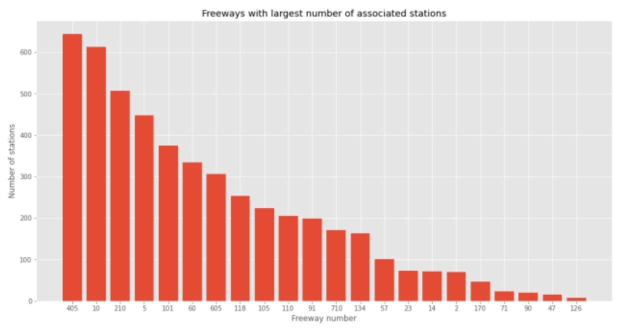


Figure VII-2 – Bar plot of freeways sorted by the number of associated sensors

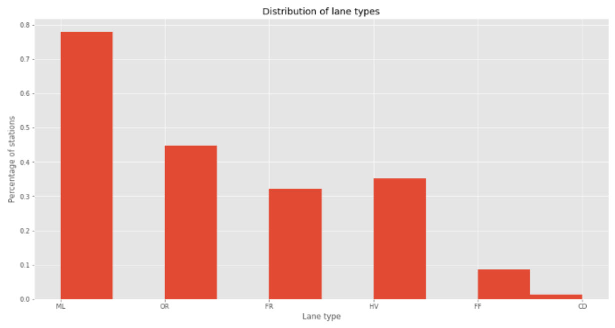


Figure VII-1 – Normalized histogram of sensors by the lane types

a major freeway in California that traverses the westside of Los Angeles from north to south. It is followed by interstate 10, which is another important freeway that connects all the south states of the United States, starting from Santa Monica, CA, until Jacksonville, FL. Given the importance of these freeways, it is clear why there are so many sensors measuring their traffic. In Figure VII-1, the distribution of lane types is displayed. Most of the sensors are placed on the mainline (ML), followed by On Ramp (OR), high occupancy vehicle lane (HV) and off ramp (FR).

Even if the number of sensors located around the city is large, not all of them work properly all the time. The percentage of correct measurements which have not been imputed by the management system are reflected in each 5-minute timestamp by the "% Observed" field. For each station, the mean of these percentages can be computed for the entire month of April to see how

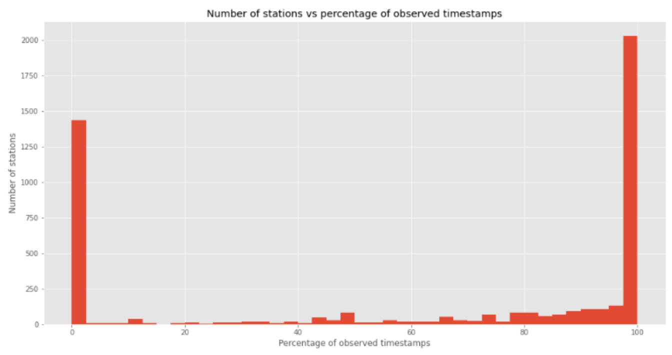


Figure VII-3 – Number of sensors having certain percentages of observed values

much data is measured. In Figure VII-3 a histogram that displays the number of stations having certain percentages of observed timestamps. Most of the station either do not measure data at all or measure it almost all the time. The number of stations that have a variable percentage is small, but still noticeable.

Another interesting plot regarding the percentage of observed values is the map one, displayed in Figure VII-4. It shows the geographical position of sensors with at least 70% mean observed values, and their color represents the value of this mean, with red being higher, maximum 100% and green being lower, minimum 70%. It can be noticed that most of the sensors are indeed red, confirming the high number of sensors with high observed percentage and that there is a noticeable number of sensors with a lower percentage. The map also displays many roads which do not have any sensors highlighted, meaning that they have done a smaller than 70% number of measurements, which can even go down to 0%. In the Figure VII-5, which displays sensors with smaller than 10% observed values, the missing sensors are highlighted.

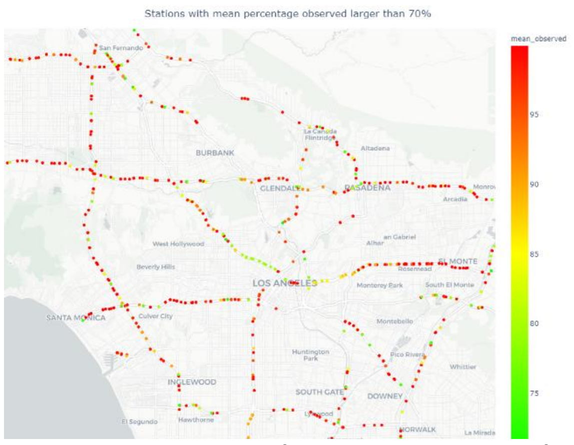


Figure VII-4 – Map plot of sensors with percentage of observed values larger than 70%

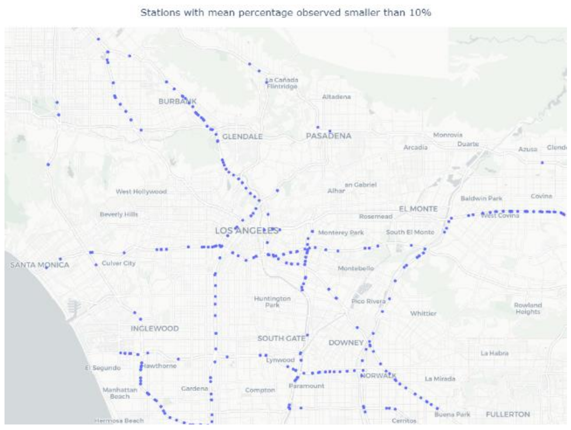


Figure VII-5 Map plot of sensors with percentage of observed values less than 10%

Because freeway I-405 has the most sensors, an analysis at the level of the freeway will be done by grouping all records at timestamp level and averaging all the flow values of the sensors which are associated with the freeway for each timestamp.

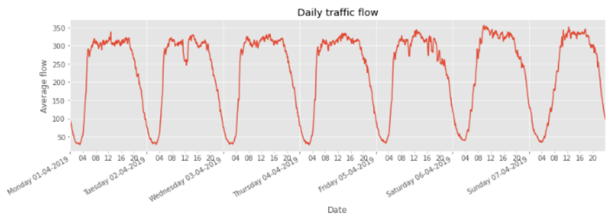


Figure VII-9 – Traffic flow values in a week for I-405

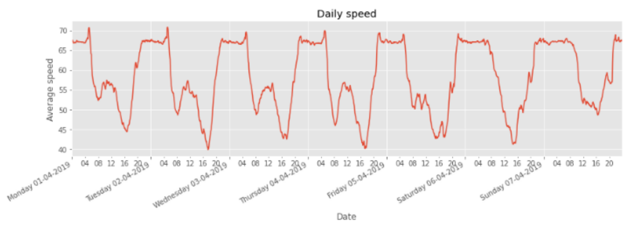


Figure VII-8 – Average speed values in a week for I-405

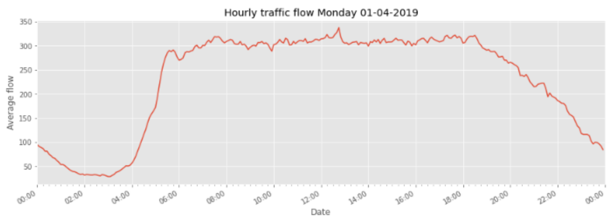


Figure VII-11 – Hourly traffic flow values in a workday for I-405

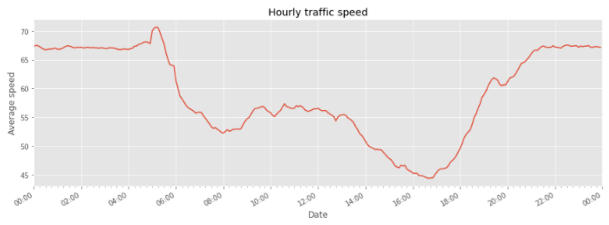


Figure VII-10 Hourly average speed values in a workday for I-405

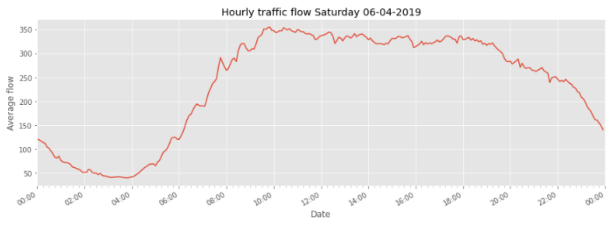


Figure VII-7 Hourly traffic flow values in a weekend day for I-405

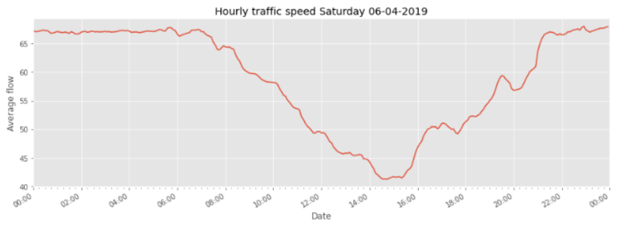


Figure VII-6 Hourly average speed values in a weekend day for I-405

In the above figures there are several plots made on data from freeway I-405. Figure VII-8 and Figure VII-9 display average traffic flow and speed values in a full week. The most noticeable insights one can gain is the periodicity of the data, each day following certain patterns, like low traffic flow during the night and high flow during the day.

Differences can be seen between slope of morning flow increase between workdays, when it is very abrupt, and weekends, when it is more relaxed. This translates into speed values, which during workdays drop in the mornings at about 8am because that is the time when most of the people go to work, and during weekends does not drop until midday.

These differences are analyzed more in-depth in the figures Figure VII-6, Figure VII-7, Figure VII-10, Figure VII-11, which display flow and speed values hourly during Monday and Saturday.

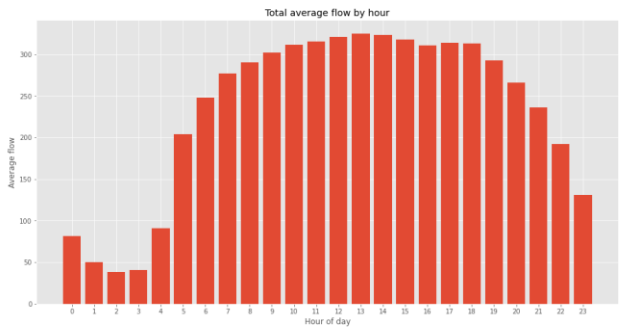


Figure VII-13 – Bar plot of average traffic flow each hour of a day

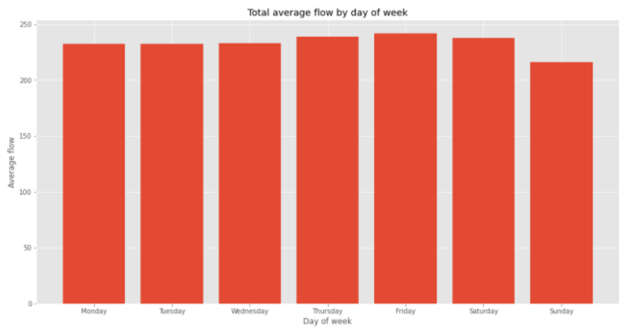


Figure VII-12 – Bar plot of average traffic flow each day of a week

By aggregating all the timestamp flow measurements and grouping them by hour and by day of week, other insights can be observed, for example least busy hour, 2AM, and most busy hour, 1PM, or least busy day, Sunday, and most busy day, Friday.

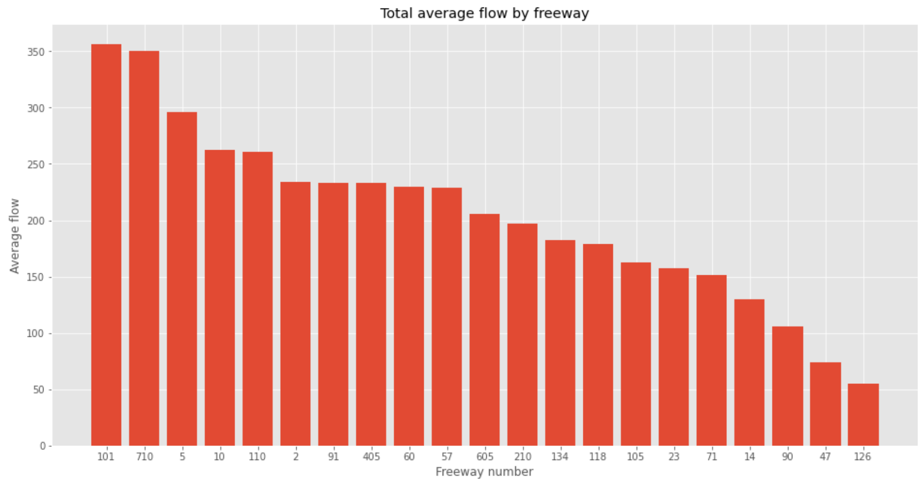


Figure VII-14 – Sorted bar plot of freeways and average flow measured on them

When it comes to roads, from Figure VII-14 it can be seen that the busiest is highway 101, which is a road that spreads from north of the country until the south along the west coastline and connects the states Washington, Oregon and California. It is followed by interstate 710, also named Los Angeles River freeway, which connects north of Los Angeles to Long Beach which is south of the city.

The relationship between flow and speed is then analyzed in Figure VII-15. This figure displays speed values on the horizontal axis and flow values on vertical axis, while the dot color is the time of day, green for night and blue for day. Night and day decision was made based on the sunrise and sunset of that day. Again, the largest values for traffic flow are during the day, but it is also the time when speed values are at their lowest, while in the night the flow may be lower, but the speed is higher. But the main point is that as the flow value increases, speed value decreases at some point due to the space limitations of the road.

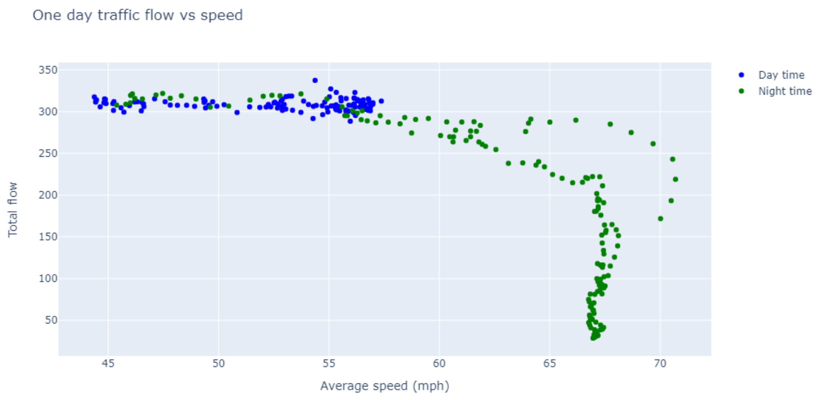


Figure VII-15 – Flow-speed plot of data measured a day on I-405

This relationship plot seems to match the one from the flow-speed model presented in Figure III-3. Low traffic flow and high-speed zone during night is the congestion free zone, while the congestion appears at high flow values when speed reaches minimum for that day. However, the theoretical queue zone in which speed is low and flow is also low does not seem to occur in this day, which means no major queues occurred on this day.

Given the points from Figure VII-15, the relationship between flow and speed can be modeled using a polynomial 2nd degree function, as displayed in Figure VII-16, in which the approximation function has the formula from Equation 7.1.

$$-1482.64 + 71.64x - 0.71x^2 \tag{7.1}$$

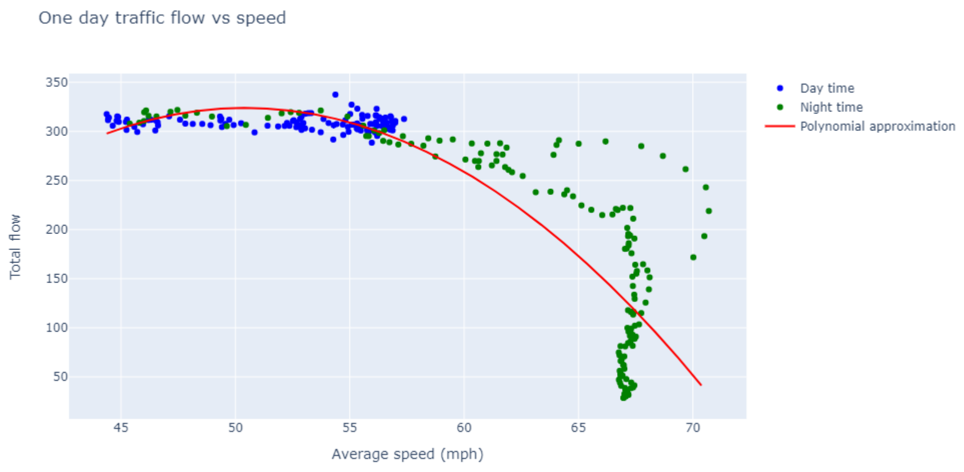


Figure VII-16 - Flow - speed polynomial approximation

The polynomial approximation has obtained an $R^2 = 0.74$ and $RMSE = 52.49$.

VII.3 Road level predictions

First series of experiments will be done at a road level, using the flow measurements of the freeway I-405 in April 2019. Flow values of all the timestamps associated with this freeway have been grouped by timestamp and averaged, resulting in average traffic flow values for each timestamp for the entire freeway. These values will be grouped into training examples of 12 input timestamps trying to output the next following timestamp from the dataset. That means, flow data from last hour (12 timestamps * 5 minutes) is used to forecast values of the traffic flow after a period of 5 minutes has passed.

When splitting the dataset, flow data from the first 2 weeks of April 2019 will be part of the training set, values of the 3rd week will be the validation set and values from the 4th week the test set. Data is normalized by considering maximum and minimum flow values and bringing all the values into the range [0,1].

The optimization algorithm is Adam, having an adaptive learning rate with an initial value of 10^{-3}, and for loss computation the mean squared error function is employed. Training is done using a constant batch size of 32 examples and 30. However, number of epochs can vary since early stopping is being used in order to stop the training process when there are no more validation loss value improvements after 6 consecutive epochs. Besides that, the learning rate of the optimizer is also changed when validation loss stops improving – its value is divided by 10 after 3 consecutive epochs of no validation loss value improvements.

VII.3.1 Time series analysis using machine learning

Two basic machine learning algorithms will be trained and evaluated on flow forecasting task: linear regression and multi-layer feedforward neural network, and one deep learning architecture specialized in sharing weights between time series data, recurrent neural network with LSTM cells.

Structurally speaking, the simple neural network has an architecture which is built using 2 intermediary layers, with each one having a number of 64 trainable nodes, with ReLU activation and Dropout of 20% chance between hidden layers and between last hidden layer and output. When it comes to the recurrent neural network, it is unidirectional and uses a stacked architecture of 2 LSTM cells, whose vector size is 64 as well and use Dropout for regularization.

The validation loss and the mean squared error values on the test set are displayed on Table VII-1. In Figure VII-17 are plotted the predictions on the test set compared between the 3 models and ground truth.

	VALIDATION LOSS	**TEST RMSE**
LINEAR REGRESSION	0.0197	23.07
DENSE MODEL	0.0005	22.63
RNN WITH LSTM	0.0013	10.36

Table VII-1 – Experimental results with different algorithms

Best results have been achieved using the recurrent neural network with LSTM cells, which was the expected result. After optimizing the hyperparameters of this architecture, we obtain an even better model, that has 2 LSTM layers of sizes 128 and 64 with an obtained validation loss of **0.0010** and test RMSE **9.37**.

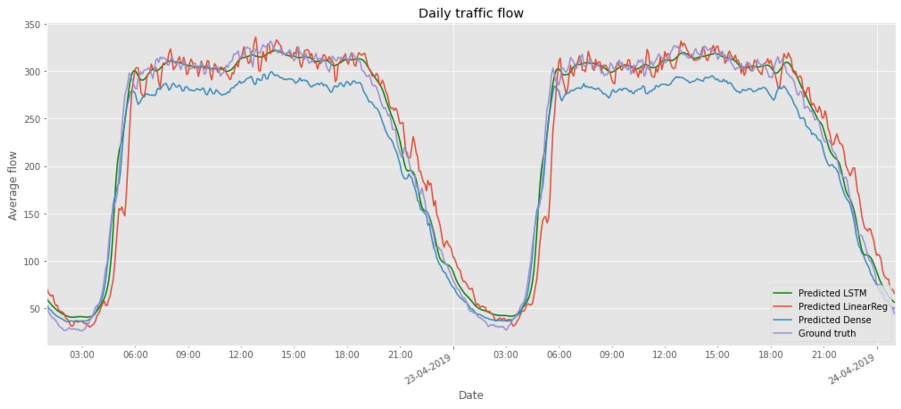


Figure VII-17 – Predicted traffic flow on test set of the three models in comparison with ground truth

VII.3.2 Custom handcrafted features

For obtaining an even improved performance, some feature engineering can be done, which is done while having in mind the kind of task which is trying to be solved. Custom features

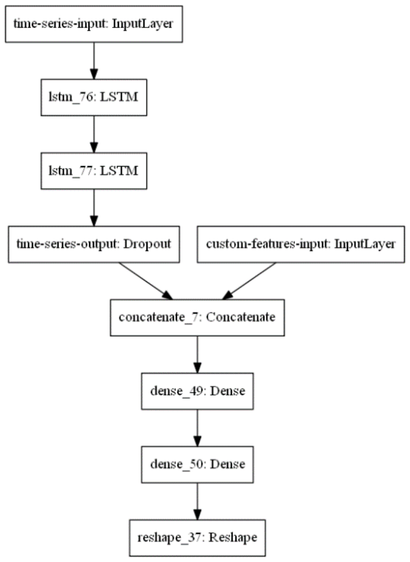

time-series-input: InputLayer
lstm_76: LSTM
lstm_77: LSTM
time-series-output: Dropout
custom-features-input: InputLayer
concatenate_7: Concatenate
dense_49: Dense
dense_50: Dense
reshape_37: Reshape

Figure VII-18 Graph of the combined model

represented by temporal indicators - hour, minute, day of week and day of month have been inserted into a model which combines them with the time series analysis model developed previously. The Keras implementation is described in Figure VII-18.

After training and evaluating the model, the performance got better, obtaining a test RMSE of **8.12**, which is an improvement over the time series only model. It means that the temporal context provides relevant information to the model. The predictions on the test set are plotted in Figure VII-19.

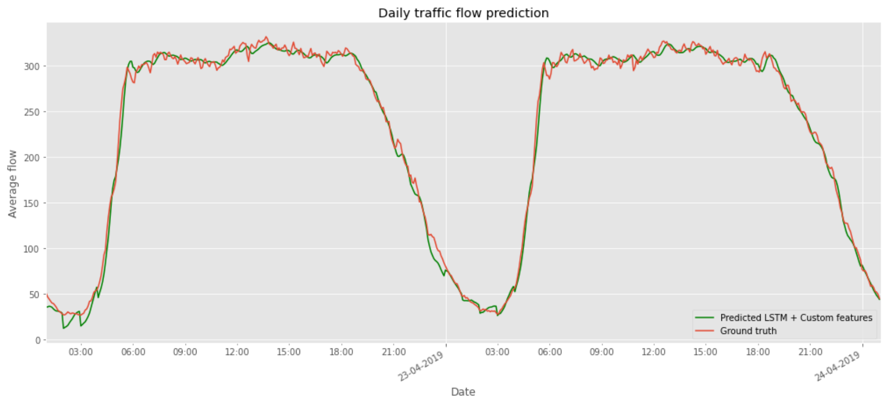


Figure VII-19 – Predictions on the test set of the combined model compared with ground truth

VII.4 Sensor level predictions

Traffic flow prediction at the level of individual sensors is another relevant analysis perspective because it gives a more granular view of the traffic situation and can be more helpful in real time scenarios. The experiments will be done at the level of an individual sensor, using the same time series analysis model as the one employed in the road level experiments, and then will be done on a network of sensors. The sensor involved in first part of the experiment is also part of the network from second part.

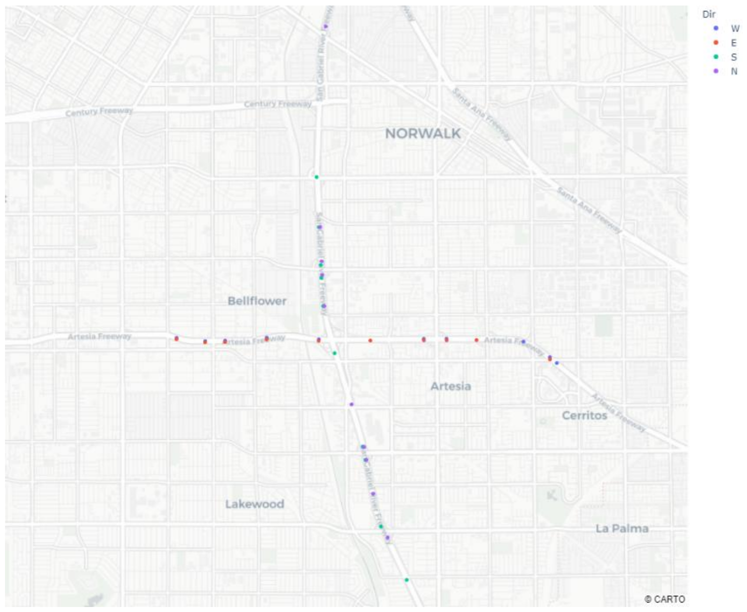


Figure VII-20 – Map plot of the manually picked sensors

Because in the data analysis chapter it has been found that there are several sensors that have values which are not measured, but imputed by the system, the first step of the experiments will be choosing sensors that have many observed values and which are closer together. Using the map plot of the sensors and their associated percentage of observed values, 40 sensors found in all directions of a 4-way interchange have been manually selected.

These sensors are displayed in Figure VII-20. Their color represents the traffic direction of the lanes in which they are found, to tell the difference between sensors found in same locations of the freeways, but in opposite lanes.

VII.4.1 Single sensor analysis

First experiment in this category is time series analysis for just one sensor, using the model previously developed for road level predictions – recurrent neural network with 2 LSTM layers of sizes 128 and 64. The chosen sensor is the last one in the left part of the plot that is associated with West direction.

Like in the previous experiments, the April 2019 data of the sensor is loaded and split into 2 weeks – 1 week – 1 week for train – validation -test. Previous 12 timestamps are used to predict next timestamp – 1 hour flow values predict data in the next 5 minutes. Again, the training is done using a constant size of 32 examples for batch size and a total number of 30 epochs, with early stopping and learning rate reducing added.

The obtained test RMSE is **38.62**, which is a larger error than the one obtained on road level prediction. This is caused by the fact that road level data is averaged over multiple sensors and has a small number of sudden variations in flow values, while single sensor data has a lot of spikes, like it can be seen in the Figure VII-21, which displays the predictions made for one station model.

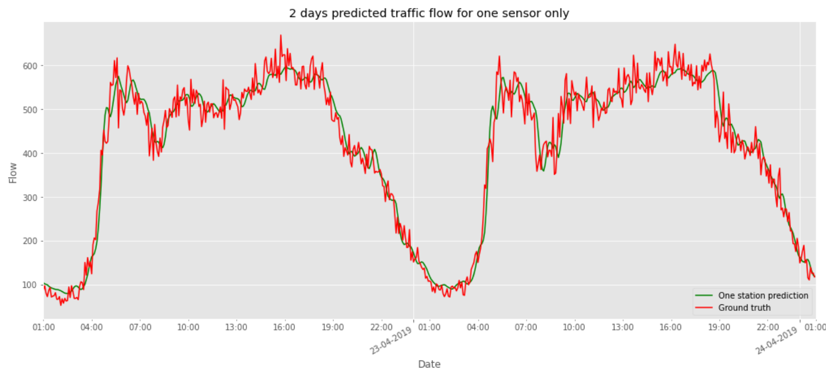


Figure VII-21 – Test set prediction on one sensor compared with ground truth

Performances of this model that is only fitted to one sensor will be compared to those of a model capable of handling all the sensors.

VII.4.2 Multiple sensors analysis

For multiple sensors prediction, a more complex model will be used, a graph neural network, which can learn the spatial dependencies between sensors and will be able to use this data for more accurate predictions for each sensor. Besides having the advantage of obtaining more accurate predictions, this kind of architecture also makes it easier for implementing the solution in a real-life scenario, since only one model can be used to achieve predictions for many sensors, instead of training one model for each individual sensor.

The model will require two inputs: a matrix of n timestamps containing flow values for all p sensors and, because sensors network is organized as a graph, an adjacency matrix that represents the connections between sensors.

The adjacency matrix is built by considering several aspects. Firstly, matrix values between sensors which are connected according to the rule described in the methodology will have a value larger than 0. Using each sensor's geographical coordinates from metadata file, the distance between connected sensors will be computed and plugged into formula from Equation 6.2, along with the parameters σ and ε and will give the weights of the matrix.

The obtained adjacency matrix is 40x40 and is represented in the following: Figure VII-22,Figure VII-23,Figure VII-24 and Figure VII-25. The plots are heatmaps of adjacency matrix weight values after using different σ values – 1, 3, 5 and 10. As weight values get closer to the maximum value, 1, the cell color gets closer to red, while zero weight values are gray. Many of the zero weights stay that way in all plots because the nodes are not considered to be connected.

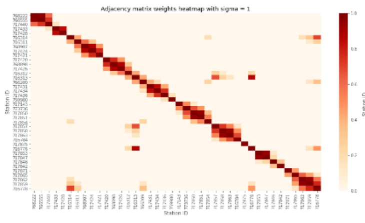


Figure VII-25 – Adjacency matrix heatmap when $\sigma = 1$

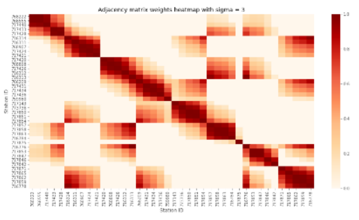


Figure VII-24 - Adjacency matrix heatmap when $\sigma = 3$

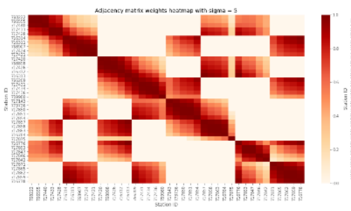


Figure VII-23 - Adjacency matrix heatmap when $\sigma = 5$

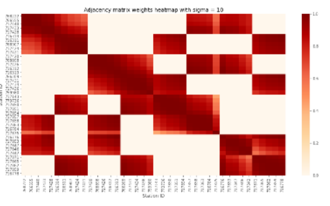


Figure VII-22 - Adjacency matrix heatmap when $\sigma = 10$

The sensors displayed in the heatmap are grouped depending on the travelling direction they measure and sorted from left side of the plot to the right side of the plot, on horizontal, and from top side of the plot to the bottom side of the plot, on vertical, in the following order: West, East, North and South.

As the value of the parameter σ increases, the weights of the matrix increase as well, with a value of 10 giving almost maximum weight values between all connections. The optimal value will be found experimentally, through training models with different values of σ used to compute adjacency matrix weights.

Besides the σ parameter used for adjacency matrix weights, other hyperparameters are used inside the model to define the architecture, which are the number of GCN layers, with the respective size of the output vector, and the number of LSTM layers, along with the size of LSTM vector output.

A grid search hyperparameter optimization will be done with the search space being described in equations (7.2), (7.3), and (7.4) and with the target metric validation loss. The initial Temporal-GCN architecture has 2 layers of GCN having variable sizes and 2 layers of LSTM with variable sizes.

$$\sigma \in \{1, 3,5,10\} \tag{7.2}$$

$$gcn\ size \in \{16,32, 64\} \tag{7.3}$$

$$lstm\ size \in \{64,128,256\} \tag{7.4}$$

The results of the training experiments are that the best validation loss have been obtained with $\sigma = 1, gcn\ size = 32, lstm\ size = 256$. Figures Figure VII-26, Figure VII-27, and Figure VII-28 show the relationship between values of searched parameters and the obtained validation loss. Validation loss is impacted by σ and $lstm\ size$, with the lowest loss values being associated with $\sigma = 1$ and $lstm\ size = 256$, while $gcn\ size$ does not seem to have a big impact on the results.

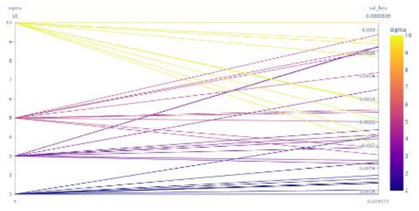


Figure VII-28 – Influence of σ on validation loss

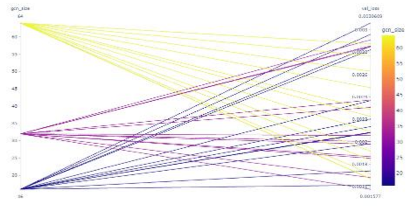


Figure VII-27 - Influence of $gcn\ size$ on validation loss

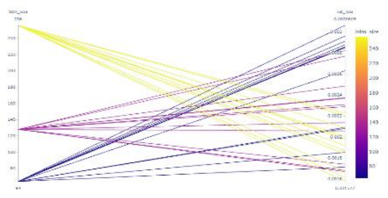


Figure VII-26- Influence of $lstm\ size$ on validation loss

The low value of σ means that only close sensors are taken into consideration and high value means that more distant are considered as well. Because the lower value is being associated with low values of validation loss it gives the conclusion that the information about sensors does not improve the accuracy, but it decreases it.

A possible reason for this might be that the number of GCN layers would need to increase as the number of connected paths in the graph increase. The next experiment will train models with different number of GCN layers and with adjacency matrices that are computed with different σ value. The validation loss values obtained during the experiment are displayed in Table VII-2

NUMBER OF GCN LAYERS	**2 LAYERS**	**3 LAYERS**	**4 LAYERS**
$\boldsymbol{\sigma = 1}$	**0.00158**	**0.00165**	**0.00158**
$\boldsymbol{\sigma = 3}$	0.00190	0.00209	0.00241
$\boldsymbol{\sigma = 5}$	0.00238	0.00281	0.00276
$\boldsymbol{\sigma = 10}$	0.00208	0.00274	0.00269

Table VII-2 – Experimental results with different number GCN layers and σ values

Again, the best results are obtained when $\sigma = 1$. Increasing the size of the GCN architecture by having a larger number of layers does not improve model performance when larger σ values are used. Since σ value is directly proportional with how tightly coupled values of the sensors network are, this results into the conclusion that the model does not have the capability to perform well when there are a lot of connections.

But the focus of this experiment is comparing the performances with those of the model that only handles one single station. The test set is passed through the best performing model which predicts data for all the sensors, out of which are extracted only predicted flow values for the sensor analyzed in the single sensor experiment. Figure VII-29 shows the output of the temporal GCN on the sensor which was previously modeled using time series analysis for one sensor only, compared with ground truth.

After comparing with the ground truth flow values of the test set, the obtained root mean squared error is **34.50**, which is an improvement over the test set error obtained when training and predicting using data from a single sensor.

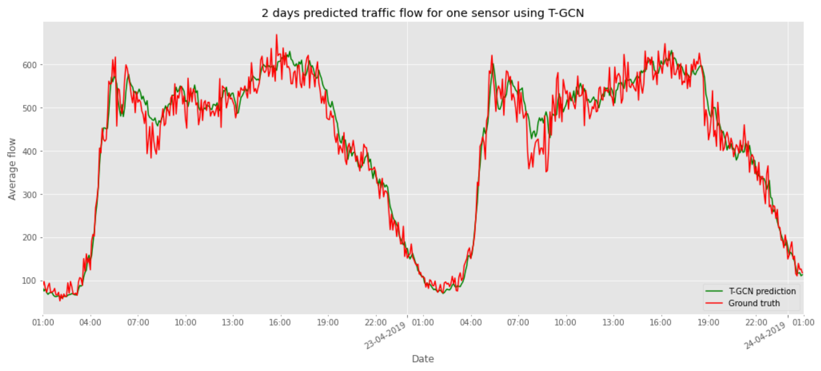


Figure VII-29 – Test set prediction of Temporal GCN compared with ground truth

The Temporal Graph Convolutional Network on multiple sensors is able to make more accurate predictions than the simple LSTM, by taking into account data from connected sensors. This is one major advantage. Another one which is just as important is the fact that using only one model there can be made predictions for 40 separate sensors, which is way better than training 40 individual models.

VIII Conclusions & future development

Traffic is a global issue which needs to be tackled such that the living conditions of the large number of people from the major urban areas to be improved. This problem led to the development of a new study domain – traffic flow theory, which uses mathematical models to describe real life scenarios. With the help of technology, measurements like the number of cars passing through certain areas and their average speed are now being done using tools like inductive loop sensors and traffic cameras. State agencies, like California Department of Transportation, took advantage of these measurement tools and built platforms, like PeMS, able to aggregate traffic data and make it available to the public. This data can be further analyzed in order for people to get important insights from it or it can be used for building models able to predict traffic data, since traffic flow values forecasting has an important role when it comes to traffic optimization.

Deep learning is a field of study which has been getting more and more popularity, due to the impressive results obtained on a wide variety of tasks using large amounts of data. Architectures like recurrent neural networks with LSTM cells have proven their capability of solving time series analysis tasks. The non-Euclidean spatial analysis using Graph Neural Networks is one of the most recent advances in this field. Graph neural network can be combined with recurrent neural networks for spatial-temporal analysis models.

This paper explores traffic data measured in the city of Los Angeles, California and uses it to train and evaluate models based on state-of-the-art deep learning algorithms on the task of traffic flow forecasting which can be used for traffic optimization.

Simple machine learning techniques were able to model time series data from road-level traffic flow values with low error. By employing deep learning algorithms, like recurrent neural networks with memory passing mechanism based on LSTM cells, the error improves even further on the test set. Even bigger improvements are observed after combining the LSTM model with handcrafted features related to current temporal context, like current day, hour, and minute.

Graphs are a suitable way of modeling spatial information regarding to traffic measurement sensors. Graph Convolutional Networks are recent deep learning architectures able to learn on such data. Combining Graph Convolutional Networks with Recurrent Neural Networks is a method for training spatial-temporal dependencies of the sensors and resulted in a model which was able to predict flow values of a sensor for 1 week with a smaller error than using just a Recurrent Neural Network trained on that single sensor's data, due to the extra information received from the other connected sensors.

The model could be improved even more by using more custom features and by training it with larger amounts of data, gathered from more days and more sensors. Subsequently, it could then be used for predicting congestions in traffic management systems that would take actions like modifying traffic lights durations and in route planning software that would build a route based on which roads are expected to have a light traffic.

IX Bibliography

[1] D. Schrank, B. Eisele and T. Lomax, "2019 Urban Mobility Report," Texas A&M Transportation Institute, 2019.

[2] "TomTom Traffic Index - Los Angeles," TomTom, [Online]. Available: https://www.tomtom.com/en_gb/traffic-index/los-angeles-traffic/. [Accessed 31 May 2021].

[3] "TomTom Traffic Index - Bucahrest Traffic," TomTom, [Online]. Available: https://www.tomtom.com/en_gb/traffic-index/bucharest-traffic/. [Accessed 31 May 2021].

[4] "TomTom Traffic Index Ranking," TomTom, [Online]. Available: https://www.tomtom.com/en_gb/traffic-index/ranking/. [Accessed 31 May 2021].

[5] C. Giarratana, "4 Ways Cities Are Using Smart Technology To Control Traffic Congestion," Traffic Safety Store, 18 February 2019. [Online]. Available: https://www.trafficsafetystore.com/blog/4-ways-cities-are-using-smart-technology-to-control-traffic-congestion/. [Accessed 2021 May 2021].

[6] N. Gartner, C. Messer and A. Rathi, "Revised Monograph on Traffic Flow Theory," Federal Highway Administration Research and Technology, 2001.

[7] H. Song, H. Liang and H. Li, "Vision-based vehicle detection and counting system using deep learning in highway scenes," *Eur. Transp. Res. Rev.,* vol. 11, no. 51.

[8] "General Data Protection Regulation," [Online]. Available: https://gdpr-info.eu/. [Accessed 6 June 2021].

[9] . T. V. Mathew, "NPTEL Transportation Engineering I," [Online]. Available: https://nptel.ac.in/content/storage2/courses/105101087/downloads/Lec-33.pdf. [Accessed 31 May 2021].

[10] S. Ruder, "An overview of gradient descent optimization algorithms," 2016.

[11] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research,* 2014.

[12] R. Baijayanta, "All about Feature Scaling," 6 April 2020. [Online]. Available: https://towardsdatascience.com/all-about-feature-scaling-bcc0ad75cb35. [Accessed 31 May 2021].

[13] "A Beginner's Guide to LSTMs and Recurrent Neural Networks," Pathmind, [Online]. Available: https://wiki.pathmind.com/lstm. [Accessed 31 May 2021].

[14] J. e. a. Zhou, "Graph neural networks: A review of methods and applications," *AI Open ,* vol. 1, pp. 57-81, 2020.

[15] C. Pham, "Graph Convolutional Networks (GCN)," [Online]. Available: https://www.topbots.com/graph-convolutional-networks/. [Accessed 07 June 2021].

[16] T. N. Kipf and W. Max, "Semi-Supervised Classification with Graph Convolutional Networks," *arXiv preprint arXiv:1609.02907,* 2016.

[17] . M. S. Ahmed and A. R. Cook, "Analysis of freeway traffic time-series data by using Box–Jenkins techniques," *Transp. Res. Rec.*, no. 722, p. 1–9, 1979.

[18] M. van der Voort, M. Dougherty and S. Watson, "Combining Kohonen maps with ARIMA time series models to forecast traffic flow," *Transp. Res. C, Emerging Technol.*, vol. 4, no. 5, p. 307–318, Oct. 1996.

[19] B. M. Williams and L. A. Hoel, "Modeling and forecasting vehicular traffic flow as a seasonal ARIMA process: Theoretical basis and empirical results," *J. Transp. Eng.*, vol. 129, no. 6, p. 664–672, Nov./Dec. 2003.

[20] I. Okutani and Y. J. Stephanedes, "Dynamic prediction of traffic volume through kalman filtering theory," *Transportation Research Part B Methodological*, vol. 18, no. 1, p. 1–11, 1984.

[21] X. L. Zhang, . H. E. Guo-Guang and L. U. Hua-Pu, "Short-term traffic flow forecasting based on k-nearest neighbors non-parametric regression," *Journal of Systems Engineering*, vol. 24, no. 2, p. 178–183, Feb. 2009.

[22] Z. S. Yao, . C. F. Shao and Y. L. Gao, "Research on methods of shortterm traffic forecasting based on support vector regression," *Journal of Beijing Jiaotong University*, vol. 30, no. 3, p. 19–22, 2006.

[23] K. Kumar, M. Parida and V. K. Katiyar, "Short term traffic flow prediction for a non urban highway using artificial neural network," *Proc. Soc. Behav. Sci.*, vol. 104, p. 755–764, Dec. 2013.

[24] R. Fu, Z. Zhang and L. Li, "Using lstm and gru neural network methods for traffic flow prediction," *Youth Academic Conference of Chinese Association of Automation*, p. 324–328, Jan. 2017.

[25] Y. Lv, . Y. Duan, W. Kang, Z. Li and . F. Y. Wang, "Traffic flow prediction with big data: A deep learning approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, p. 865–873, Jan.2015.

[26] Y. Wu and H. Tan, "Short-term traffic flow forecasting with spatialtemporal correlation in a hybrid deep learning framework," Dec. 2016.

[27] Yu, Bing, Yin, Haoteng, Zhu and Zhanxing, "Spatio-temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[28] L. Zhao, Y. Song, C. Zhang, Y. Liu, P. Wang, T. Lin, M. Deng and H. Li, "T-GCN: A Temporal Graph Convolutional Network for Traffic Prediction," *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[29] "Caltrans Performance Measurement System (PeMS)," California Deprtament of Transportation, [Online]. Available: https://pems.dot.ca.gov/. [Accessed 6 June 2021].

[30] Google Brain, "TensorFlow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation*, 2016.

[31] F. Chollet, "keras," 2015. [Online]. Available: https://github.com/keras-team/keras. [Accessed 22 iunie 2019].

[32] CSIRO's Data61, "StellarGraph Machine Learning Library," *GitHub Repository*, 2018.

[33] V. Mahendran, "Recurrent Neural Networks," 1 March 2019. [Online]. Available: https://towardsdatascience.com/recurrent-neural-networks-d4642c9bc7ce. [Accessed 31 May 2021].

[34] C. Olah, "Understanding LSTM Networks," 27 August 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed 31 May 2021].