Evolutionary neural architecture search for the task of road traffic prediction.

Daniel Klosa

A thesis presented for the degree of
Doctor of Engineering

AG Optimierung und optimale Steuerung
Zentrum für Industriemathematik
Universität Bremen
11.04.2024

Promotionskomitee:

1. Gutachter: Prof. Dr. Christof Büskens
2. Gutachter: Prof. Dr. Dr. h.c. Peter Maaß

Datum des Promotionskolloquiums: 5. Juli 2024

Kurzreferat

Das Thema dieser Dissertation ist die Anwendung der evolutionären neuronalen Architektursuche, um geeignete neuronale Netze für die Vorhersage von Geschwindigkeit und Verkehrsfluss aus Straßenverkehrsdaten zu finden. Die Arbeit beginnt mit der Beschreibung der Messdaten und der Beschreibung des Vorhersageproblems. Anschließend werden grundlegende Konzepte aus den Bereichen Maschinelles Lernen, Deep Learning und Neuronale Architektursuche (NAS), insbesondere hinsichtlich der Anwendung, erläutert. Der letzte Teil dieser Dissertation besteht aus fünf Artikeln, zu denen der Autor dieser Arbeit einen wesentlichen Beitrag geleistet hat.

Die ersten beiden Artikel geben einen Überblick über das Problem der Verkehrsdatenvorhersage anhand von Messdaten aus der Stadt Bremen. Es wird die Maschinelles Lernen Methode k-nearest neighbours vorgestellt und auf die Messdaten angewandt. Zusätzlich wird die Verbesserung des Modells durch den Einsatz von Datenfüllmethoden untersucht. Im dritten Artikel vergleichen wir kombinierte polynomiale Regressionsmodelle, ein einfaches Maschinelles Lernen Modell, mit graphischen neuronalen Netzen. Diese Art von Netzen enthalten spezielle Operationen, die räumliche Abhängigkeiten zwischen Messpunkten berücksichtigen. Die Anwendung evolutionärer neuronaler Architektursuche wird im vierten Artikel vorgestellt. Das Ergebnis des in unserem Framework verwendeten genetischen Algorithmus hängt von der Fitness, d. h. der Kostenfunktion auf dem Datensatz, jeder Architektur im Suchraum ab. Die Wahl des Validierungsverlustes als Fitness ist zwar ideal im Hinblick auf die Genauigkeit, verlangsamt aber den Algorithmus enorm, da er das Training der neuronalen Netze bis zur Konvergenz erfordert. Um die Verwendung unseres Frameworks praktikabel zu machen, evaluieren wir daher im fünften Artikel Zero-Cost Proxies, die eine Fitness für Architekturen auf der Grundlage von einzelnen Vorwärts- oder Rückwärtsdurchläufen durch das Netzwerk berechnen. Die Evaluierung pro Netzwerk dauert daher nur wenige Sekunden statt mehreren Stunden. Wir zeigen, dass der naswot Zero-Cost Proxy robust gegenüber zufälligen Initialisierungen der Gewichte, Netzwerk- und Batchgrößen ist und eine hohe Spearman-Rank-Korrelation mit dem Validierungsverlust aufweist.

Mein Beitrag ist ein Framework für die Suche nach neuronalen Architekturen, die besonders geeignet für die Vorhersage von Verkehrsdaten sind. Mein NAS-Framework findet auf einem gegebenen Datensatz eine Architektur, die mit von Experten designten neuronalen Netzwerken und neuronalen Netzwerken, die von anderen NAS-Frameworks gefunden wurden, in Bezug auf Kostenfunktion und Rechenzeit mithalten oder diese übertreffen kann.

Abstract

The topic of this dissertation is the application of evolutionary neural architecture search to find suitable neural networks for predicting speed and flow from road traffic data. The thesis begins by describing the measurement data and describing the forecasting problem. Following this, fundamental concepts in the fields of Machine Learning, Deep Learning, and Neural Architecture Search (NAS), particularly concerning application, are explained. The last part of this dissertation consists of five articles to which the author of this thesis has made a significant contribution.

The first two articles provide an overview of the problem of traffic data prediction, concerning measurement data from the city of Bremen. The machine learning model k-nearest neighbors is introduced and applied to the measurement data. In addition, we evaluate data imputation methods to improve models. In the third article, we compare combined polynomial regression models, a simple machine learning model, with graph convolutional neural networks. These are neural networks that include special operations incorporating spatial dependencies between measurement points. Our evolutionary neural architecture search framework is presented in the fourth article. The outcome of the genetic algorithm used in our framework depends on the fitness, i.e. performance on the dataset, of each architecture in the search space. While the choice of validation loss as fitness is ideal w.r.t. the accuracy, it slows down the algorithm tremendously since it necessitates training the neural networks until convergence. Hence, to make usage of our framework viable, in the fifth article, we evaluate zero-cost proxies, which compute a fitness for architectures based on singular forward or backward passes through the network. Therefore, evaluating network fitness only takes a few compared to multiple hours. We show that the naswot zero-cost proxy is robust w.r.t. random initializations of weights, network sizes and batch sizes and has a high spearman rank correlation with the validation loss.

My contribution is a neural architecture search framework that finds neural network architectures that are especially powerful for predicting road traffic data. My NAS framework finds an architecture for a given dataset that can keep up with or outperform handcrafted neural networks and neural networks found by other NAS frameworks in terms of performance and computation time.

Danksagung

Ich habe diese Dissertation während meiner Zeit als wissenschaftlicher Mitarbeiter in der AG Optimierung und optimale Steuerung (O2C) am Zentrum für Industriemathematik der Universität Bremen geschrieben. Dies geschah im Rahmen des $DiSCO_2$ Projektes, in dem es um die Straßenverkehrsvorhersage in Bremen ging. Dabei durfte ich meine Stärken im Deep Learning und der Neuronalen Architektursuche anwenden, obwohl diese nicht zu den eigentlichen Themengebieten der O2C gehören. Mein besonderer Dank gilt daher meinem Doktorvater Prof. Dr. Christof Büskens, der mir diese Möglichkeit gegeben hat und meine Arbeit stets mit viel Verständnis unterstützt und gefördert hat. Die zahlreichen Gespräche auf intellektueller und persönlicher Ebene werden mir immer als bereichernder und konstruktiver Austausch in Erinnerung bleiben.

In die Richtung des Deep Learning hat mich Prof. Dr. Dr. h.c. Peter Maaß 2017 gelenkt, als er mir während einer Zugfahrt neuronale Netze erklärt hat. Für mich war dies ein sehr prägender Moment, und daher bin ich sehr stolz, dass er meine Doktorarbeit als zweiter Gutachter betreut hat.

Ganz besonders bedanken möchte ich mich bei meiner Betreuerin Malin Lachmann. Von der Unterstützung in den Projekten, die ich in der AG bearbeitet habe, und bei meiner Dissertation, sei es durch die vielen geplanten und ungeplanten Meetings, sowohl inhaltlich als auch organisatorisch, über ihre Fähigkeiten mich zu motivieren meine Arbeit voran zu bringen, bis hin zu ihrer Unterstützung bei persönlichen Angelegenheiten. Auf Malin konnte ich mich immer verlassen.

Ebenfalls bedanken möchte ich mich bei meinen Arbeitskollegen. Amin Mallek hat mich über Jahre mit wissenschaftlichem Input unterstützt, zusammen mit mir Forschung im $DiSCO_2$ Projekt betrieben, mit mir wissenschaftliche Artikel veröffentlicht und meine Dissertation Korrektur gelesen. Felix Langen hat zu einigen Arbeiten im $DiSCO_2$ Projekt beigetragen, meine Dissertation Korrektur gelesen und Forks beim Schachspielen angesagt. Tekwa Tedjini und Lennart Evers haben mir wertvollen wissenschaftlichen Input zu meinen Veröffentlichungen gegeben. Viele andere Mitarbeiter aus der AG Optimierung und optimale Steuerung haben mir wertvolles Feedback bei meinen Präsentationen im Research Seminar gegeben.

Insbesondere möchte ich mich bei meiner Frau Mary Markley bedanken, die mich zu Hause immer unterstützt hat, mir die Zeit gegeben hat an meiner Dissertation zu arbeiten und viele meiner Englisch-Fehler in dieser Arbeit korrigiert hat. Auch bedanken möchte ich mich bei meiner Tochter Everlee für das häufige Ablenken von der Arbeit.

Mein besonderer Dank gilt schließlich neben vielen Freunden und der Familie meinen Eltern Harald und Ute, die mich mein gesamtes Leben in allem unterstützt haben. Meinem Vater habe ich meine experimentierfreudigkeit und technisches Wissen zu verdanken. Er ist in diesem Jahre verstorben und ich widme ihm diese Arbeit.

Diese Arbeit wurde durch den Europäischen Fonds für regionale Entwicklung Bremen (EFRE) gefördert.

Contents

1 An introduction to the task of road traffic prediction

40 hours – the time every driver in Bremen lost in 2022 due to traffic congestion [31]. Road traffic is a complex and sensitive system, especially in large cities. Small jams due to accidents, construction sites or sub-optimally programmed traffic lights can grow quickly due to the high density of vehicles. They lead to congestions in the whole city due to travellers adjusting their routes accordingly. A jam in one region can therefore cause further jams in other regions. With an increasing number of cars on the roads, the only chance of alleviating this problem is by employing intelligent transportation systems (ITS) [86]. These systems can help stakeholders make coordinated, safe and smart decisions when it comes to traffic management. To achieve this, it is essential to gather, analyse and predict data on various parameters such as speed and flow of vehicles on the road.

Speed data refers to the velocity at which vehicles move on the road. It is an important parameter that influences traffic flow and safety. Accurate speed data can help in predicting the travel time of vehicles and identifying congestion points. On the other hand, flow data refers to the number of vehicles that pass through a particular point on the road during a given time period. It is a critical parameter in understanding the overall traffic volume on a road and can help in identifying congestion points and bottlenecks.

To gather such data, various methods and technologies are used. One common method of collecting speed and flow data is through the use of induction loops. Induction loops are wires embedded in the road surface that are used to detect the presence of vehicles. These loops work by generating a magnetic field that is disturbed by the presence of a vehicle, allowing the system to detect the passage of a vehicle and measure its speed. However, speed measurements are only accurate when a measurement site has two induction loops not far apart. If this is not the case, speed is often computed by dividing the length of the loop detector by the time a vehicle was present. It can easily be seen that this method will lead to inaccurate data.

In order to record measurement data, induction loops have to be installed in specific locations on the road. The loops are typically placed where drivers have to take decisions, e.g. before and after junctions or on and off ramps. The loops are then connected to a data collection unit that records the data and sends it to a central system for

analysis. In Bremen, the data gets sent to the Verkehrsmanagementzentrale (VMZ) and is used for monitoring traffic. During the research project DiSCO$_2$ (data-based and intelligent simulation of traffic for CO_2 reduction in Bremen) we have worked on a hand selected subset of historical data from the urban road network of Bremen [39, 55, 56]. The dataset consists of seven detectors on a junction over the span of 11 weeks. We accumulated traffic flow data into 10 min intervals, resulting in 11088 timesteps. The specifics are shown in Table 1.1. We additionally use datasets from California, which are commonly used in the literature to evaluate the methods developed in our work on neural architecture search [37, 38]. The California Department of Transportation (Caltrans) has made its recordings of traffic publicly available. Measurements are obtained from the Performance Measurement System (PeMS) of Caltrans. In this work we focus on four of these data subsets, which are summarised in Table 1.1.

Table 1.1: Overview of datasets used in this work.

Dataset	Type	Region	Timeframe	#timesteps	#sensors
Bremen	flow	Bremen Hbf	Apr – Jun 2018	11088	7
PeMSD4	flow	San Francisco Bay	Jan – Feb 2018	16992	307
PeMSD8	flow	San Bernardino	Jul – Aug 2016	17856	170
METR-LA	speed	Los Angeles	Mar – Jun 2012	34272	207
PEMS-BAY	speed	California Bay	Jan – May 2017	52116	325

For all of these datasets, the measurements are accumulated in 5 min intervals. The PeMSD4 dataset published by Bai et al. [4] is concerned with flow measurements from the San Francisco Bay Area, recorded from January to February in 2018. This results in 16992 timestamps for all 307 sensors. The PeMSD8 dataset [4], also concerned with flow measurements, includes 170 sensors located in the San Bernardino Area and was recorded from July to August 2016, resulting in 17856 timestamps. The METR-LA dataset published by Li et al. [43] includes speed measurements of 207 detectors within the Los Angeles County, recorded from March to June 2012 for a total of 34272 timestamps per detector. The PEMS-BAY dataset [43] also includes speed measurements and was recorded in the California Bay Area from January to May 2017. There are 325 sensor locations with 52116 timestamps each.

In Figure 1.1 example data from the PeMSD8 dataset can be seen. The flow, measured in vehicles per hour is shown for a timeframe of 27 hours for one detector. The measurements from the detector on the left show a common pattern for the traffic flow. The first data points concern the night and early morning hours when there are not as many vehicles on the road and, hence, the traffic flow has low values. Over the morning hours traffic slowly rises as people start driving to work. Usually, there is a peak in the morning before a small dip during lunch hours. Afterwards, traffic flow rises again until there is a second peak in the evening as people drive back home. Finally, the flow decreases again until the night. As mentioned, most flow curves follow this pattern, however, depending on the location, e.g. industrial or residential district, they can alter

as can be seen on the right in Figure 1.1. Here, the peak in the morning occurs earlier and is more pronounced. There is a larger dip afterwards and a smaller peak in the evening.

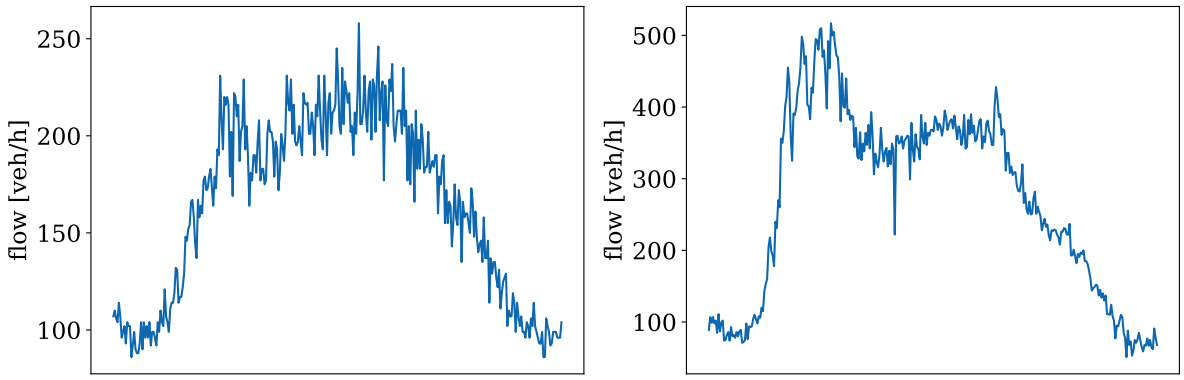


Figure 1.1: The flow in vehicles per hour from two measurement locations in the PeMSD8 dataset over a period of 27 hours.

In Figure 1.2 a whole week of data from one sensor is shown. To focus on the pattern of the flow, a Gaussian smoothing with kernel size 5 is applied. It can be seen that the traffic on each week day follows a similar pattern. Usually, Fridays differ from the rest of the weekdays. In this case, there was a high peak in flow in the evening after a dip at the preceding timestamps. This can indicate a traffic jam that is clearing up again. On weekends, especially on Sundays, there are less cars on the roads which leads to lower flow values.

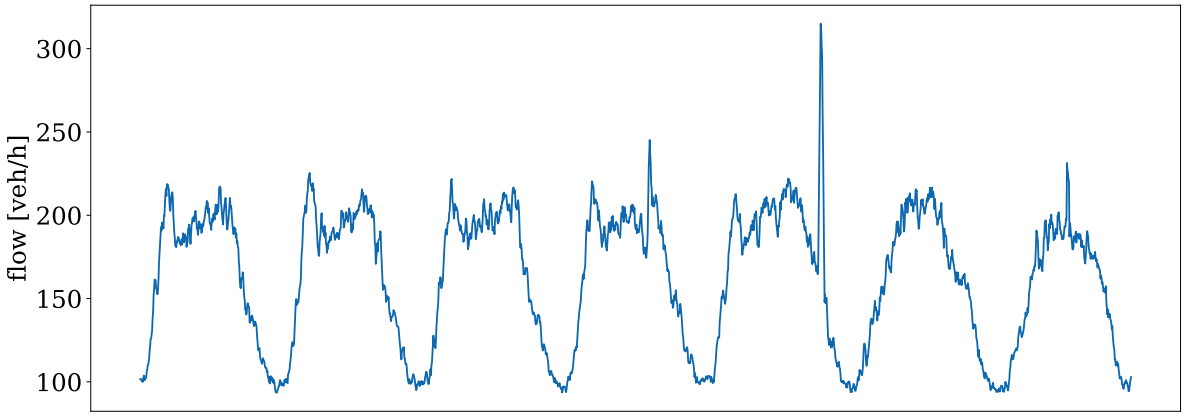


Figure 1.2: The flow in vehicles per hour from one measurement location in the PeMSD8 dataset over a period of one week starting with a Monday.

In Figure 1.3 the Monday data for seven weeks is shown. As before, a Gaussian kernel with size 5 is applied to emphasize the pattern visually. It can be seen that the flow throughout the weeks and on the same weekday generally is similar. Again, there is a

large peak in the fifth week, indicating a traffic jam clearing up. Apart from that, there are no strong dissimilarities throughout the different weeks.

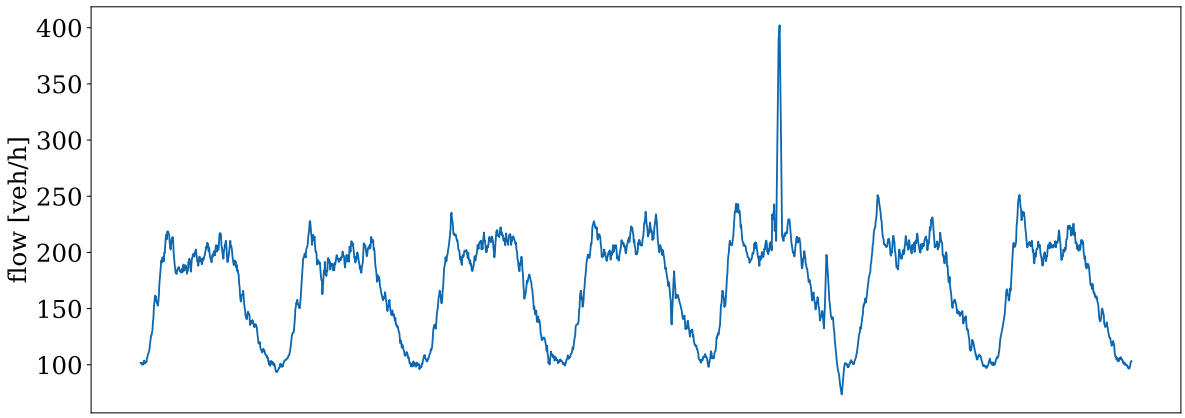


Figure 1.3: The flow from one measurement location in the PeMSD8 dataset for 7 consecutive mondays.

In addition to the dependency on time for the individual detector data, the system also has a spatial component. On their way through the road network, the same cars will pass multiple measurement locations. In Figure 1.4 a subset of detectors from the METR-LA dataset is shown. As one can see, a car driving along the freeway will pass many detectors. This results in a spatial dependency of the measurements. Mathematically, we can describe the traffic network and the detectors as a graph. To this end, let

$$\mathcal{G} = (V, E)$$

denote the graph with vertices, or, in our context, detectors $V, |V| = N \in \mathbb{N}$ and edges or connections E. To describe the relations between detectors on the edges, we use an adjacency matrix. There are different types of adjacency matrices used in road traffic networks, each offering unique insights into traffic patterns.

The binary adjacency matrix A indicates the presence or absence of edges between detectors. If there is a direct connection between detectors i and j, the corresponding entry a_{ij} is 1; otherwise, it is 0.

$$a_{ij} = \begin{cases} 1 & \text{if there is a connection between detectors } i \text{ and } j \\ 0 & \text{otherwise} \end{cases}$$

When engineering such a matrix, one could determine the number of turns a driver has to make to reach one detector from another and use a maximum number of turns as

Figure 1.4: Detector locations from the METR-LA dataset (marked as blue dots).

threshold to describe the connectivity. Similarly, there could also be a threshold for the distance between detectors or a combination of both.

The distance-based adjacency matrix D incorporates the driving distances between detectors into the graph representation. Instead of binary values, the entries d_{ij} represent the driving distances between detectors i and j.

$$d_{ij} = \text{driving distance between detectors } i \text{ and } j$$

The adjacency matrix can also be based on measurement data, e.g. traffic intensity. The weighted adjacency matrix W assigns weights to edges, representing the intensity of traffic flow between detectors. The entry w_{ij} indicates the weight of the edge between detectors i and j.

$$w_{ij} = \text{traffic intensity between detectors } i \text{ and } j$$

Traffic intensity refers to the measure of traffic flow or volume between two detectors. It quantifies the amount of vehicular movement or activity between the detectors over a specific period of time.

Traffic intensity can be measured in various ways, such as the number of vehicles passing through a particular section of the road per unit of time (e.g., vehicles per hour), the vehicle occupancy (percentage of road space occupied by vehicles), or any other suitable metric that reflects the traffic volume between detectors accurately. However, in reality, to obtain the intensity between all detectors, we would have to be able to discern where exactly individual cars are driving. The data recorded by loop detectors does not include this information.

Traffic patterns can vary based on the time of day. Time-dependent adjacency matrices capture these temporal changes in traffic flow. For example when using connectivity matrices, let each entry $a_{ij}(t)$ represent the presence or absence of a connection between detectors i and j at time t:

$$a_{ij}(t) = \begin{cases} 1 & \text{if there is a connection between detectors } i \text{ and } j \text{ at time } t \\ 0 & \text{otherwise} \end{cases}$$

Changes in the adjacency matrix could occur due to construction sites, public events or traffic jams. If this data is available, using time dependent adjacency matrices could lead to better predictive performance of the models or help in scenario simulations, e.g. simulating the changes in traffic when a construction site is opened.

The choice of which adjacency matrix to use in a model depends on the availability of data. Connectivity and driving distance based adjacency matrices are easier to compute, since they only require sensor locations and knowledge about the road network, which is often publicly available. Using a connectivity matrix poses the question of how to define connectivity of two sensor locations. We have conducted several experiments where connectivity was defined by a maximum number of turns between sensors, by a maximum driving distance and combinations. The results of those experiments were inferior when compared to using driving based matrices. For the datasets presented above, we do not have time-dependent road network information available. Therefore, in this work we concentrate on using a single driving distance based adjacency matrix for each dataset. This is also the approach most commonly used in the literature [4, 20, 43].

1.1 Traffic prediction

Accurate traffic prediction has significant implications for urban planning, transportation management and public welfare:

- Traffic flow optimization: Predictive models assist traffic management systems in optimizing signal timing and route planning, thereby reducing congestion and improving overall traffic flow.
- Emission reduction: By predicting traffic patterns, authorities can implement eco-friendly measures such as traffic diversions, reducing emissions and promoting environmental sustainability.
- Resource allocation: Efficient prediction enables better allocation of resources such as law enforcement personnel and emergency services, ensuring quick responses to accidents and traffic incidents.
- Enhanced public safety: Timely warnings to drivers about potential congestion or hazardous conditions contribute to safer road experiences, reducing the risk of accidents.

The VMZ currently uses a historic average model to predict future traffic behaviour for usage in traffic signal phase planning. They designed multiple signal phase plans, which are used depending on the time of the day and the current traffic situation. They lack forecasts of short-term (up to one hour) traffic development. While there is a good understanding of the general traffic patterns, there is no model for future traffic when it delineates from the norm. Since events like accidents can lead to vastly different behaviour in urban traffic networks than foreseen and are not predictable, short-term predictions can greatly enhance traffic management systems. Forecasting for up to an hour into the future is sufficient from an application point of view. When designing a prediction model, it has to be tuned with the available historical data. Often, this historical data comprises the last few timestamps of data, e.g. the last hour of traffic data. It can also include data about traffic jams, construction sites, weather et cetera. To this end, we define the task of traffic prediction as follows:

Definition 1 (Traffic prediction) *Let $x_t \in \mathbb{R}^{N \times F}$ denote the traffic conditions at time t on the graph of the road network $\mathcal{G} = (V, E)$ with vertices $V, |V| = N \in \mathbb{N}$ and edges E. Furthermore, let $W \in \mathbb{R}^{N \times N}$ denote an adjacency matrix. Then, given a timestamp t_0, we want to find a function $f : \mathbb{R}^{T \times N \times F} \mapsto \mathbb{R}^{T' \times N \times D}$ that predicts the next $T' \in \mathbb{N}$ timestamps of data from the last $T \in \mathbb{N}$ timestamps of data:*

$$y_{t_0} = [x_{t_0+1}, \ldots, x_{t_0+T}] = f([x_{t_0-T'+1}, \ldots, x_{t_0}]; \mathcal{G}) \in \mathbb{R}^{T' \times N \times D}$$

Here, $F \in \mathbb{N}$ and $D \in \mathbb{N}$ respectively denote the number of input and output features.

To solve the task of predicting future traffic conditions, we investigate various techniques in this thesis:

- Classical machine learning algorithms: Classical machine learning algorithms, such as polynomial regression and k-nearest neighbours are widely used to capture intricate patterns in traffic data and make accurate predictions.
- Graph neural networks (GNNs): GNNs are particularly effective for modelling spatial dependencies in road networks. They can learn complex relationships between traffic nodes (intersections or detectors) and improve prediction accuracy. Our neural architecture search framework will output GNNs that are tailored to the dataset at hand.
- Neural architecture search (NAS): NAS is an automated technique for exploring optimal neural network architectures. In traffic prediction, NAS algorithms search through a vast space of possible neural network structures, identifying architectures that are well-suited for capturing complex traffic patterns. This method significantly accelerates the development of efficient and accurate prediction models on the given dataset. NAS frameworks can have long computation times as neural networks are trained until convergence to obtain a ranking of architectures w.r.t. performance for selection. We integrate zero-cost proxies into the algorithm which estimate the performance of the network with only one forwards or backwards pass, speeding up the whole framework execution by four magnitudes. This makes it viable to use in traffic management centrals.

This leads to our main research question:

> Can we design a neural architecture search method that performs on-par or outperforms hand crafted deep learning models and classical machine learning models on various traffic flow and speed datasets?

1.2 Structure of the thesis

This thesis comprises two parts. The first one lays out the preliminaries for the papers in the second part. In this Chapter, we introduced traffic prediction in urban road networks. In Chapter 2, the mathematical foundations of machine learning and deep learning are described. We briefly describe the two classical machine learning algorithms used in the published papers, polynomial regression and k-nearest neighbours. Afterwards, we present a thorough understanding of neural network layers. In Chapter 3, neural architecture search is motivated and different ideas are described. We describe the three main components: the search space, search method and performance estimation of candidate solutions. In Chapter 4, we summarize the authors' contributions and papers. Chapter 5 focuses on summarizing these contributions and outlining future research directions. All of our publications are included in the appendix.

The author includes all five papers he wrote during his Ph.D. [37, 38, 39, 55, 56] since they are related to the task of traffic forecasting, deep learning and neural architecture search and building upon one another. It follows a list of the included papers and descriptions of the authors' contributions.

- A. Mallek, **D. Klosa** and C. Büskens: *Enhanced K-Nearest Neighbor Model For Multi-steps Traffic Flow Forecast in Urban Roads*[55]
  My contributions are Section II.2) (Enhanced KNN) including implementation of the model and Section III. (Data Description) and I had equal contribution to the Section IV. (Experimental results).
- A. Mallek, **D. Klosa** and C. Büskens: *Impact of Data Loss on Multi-Step Forecast of Traffic Flow in Urban Roads Using K-Nearest Neighbors*[56]
  My contributions are Section 2.2 (Enhanced KNN), Section 4 (Data and Reconstructed Data) and Section 5.2 (Under Artificial Datasets).
- **D. Klosa**, A. Mallek and C. Büskens: *Short-Term Traffic Flow Forecast Using Regression Analysis and Graph Convolutional Neural Networks*[39]
  My contributions are Section III.B. (Graph Convolutional Neural Networks), Section IV. (Data Description) and the implementation, experiments and discussion of the Graph Convolutional Neural Networks in Section V. (Results and Discussion).
- **D. Klosa** and C. Büskens: *Evolutionary Neural Architecture Search for Traffic Forecasting*[37]
  All parts of the paper are my contribution.
- **D. Klosa** and C. Büskens: *Low Cost Evolutionary Neural Architecture Search (LENAS) Applied to Traffic Forecasting*[38]
  All parts of the paper are my contribution.

2 Mathematical foundations

Machine learning describes the ability of a machine to learn from data. But what do we mean by that?

> *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*
>
> – Tom Mitchell, 1997 [59]

A lot of tasks T, experiences E and performance measures P are imaginable, but in this work we will concentrate on the ones specific for the task of traffic prediction.

The task of traffic prediction is one of regression. As described in Definition 1, we want to find a function $f : \mathbb{R}^{T\times N\times F} \to \mathbb{R}^{T'\times N\times D}$ that maps the graph signal X_t from a dataset $\mathcal{D}$ to the prediction $\hat{y}_t$ of the measurement y_t, i.e. $f(x_t) = \hat{y}$. In our case, the features of the input data can be the flow and speed at previous timestamps, weather data and others. The features of the outputs are our target variables, in this case, flow or speed. Hence, our dataset can be written as

$$\mathcal{D} = \{(X_t, y_t) | X_t = [x_{t-T}, \dots, x_{t-1}], y_t = [x_t, \dots, x_{t+T'-1}], \forall t = 1, \dots, m \in \mathbb{N}\},$$

where m is the number of timesteps in the dataset $\mathcal{D}$.

To be able to evaluate a machine learning model, we need to design a measure for its performance. The performance measure P has to be chosen based on the task T. Since we are focusing on regression, we use measures that compute the differences or the distances between the predictions and the measurements. The mean absolute error (MAE) is defined as

$$\text{MAE} = \frac{1}{m}\sum_{i=1}^{m} |y_i - \hat{y}_i|,$$

where y_i and $\hat{y}_i$ denote the actual and predicted values, respectively, and m represents the number of observations. The MAE provides a linear penalty for prediction errors and is particularly robust to outliers due to its absolute nature.

In contrast, the mean squared error (MSE) is given by:

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

By squaring the prediction errors, the MSE accentuates the impact of larger deviations. Since this metric is differentiable everywhere, it is preferred for gradient-based optimization techniques.

Building on this, the root mean squared error (RMSE) is essentially the square root of the MSE, thus:

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2}$$

The RMSE has the same nature as MSE while expressing the error in the same units as the target variable, aiding in interpretability.

Lastly, the mean absolute percentage error (MAPE) offers a relative measure of error and is expressed as:

$$\text{MAPE} = \frac{100\%}{m}\sum_{i=1}^{m}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

Being a percentage, the MAPE facilitates error interpretation across different scales or units, ideal for datasets with varying magnitudes.

When we evaluate a machine learning model, we are interested in its performance on data it has not seen before. To this end we usually split the dataset into three datasets. The training dataset is used to learn, i.e. tune the weights of the model. The validation dataset is used to configure possible hyperparameters of the machine learning model. This is done by evaluating the model on multiple runs with different hyperparameter configurations on the validation dataset after training and comparing the performance. The test dataset that is never shown in the training and validation process and can be

used to evaluate the final performance of the model. It is important that the test dataset can never be used for any tuning of the machine learning model.

There are two broad categories for gaining experience E. These are unsupervised and supervised learning. In unsupervised learning the data does not have labels used for classification or regression. The dataset contains many features from which we want to learn certain properties of the data like probability distributions that generated the data. The data can also be clustered as a means of learning from it. In supervised learning on the other hand, the data includes the outcome (e.g. real measurement, label) we want the machine learning model to learn. Traffic prediction is a supervised learning problem, where the target values are future flow or speed measurements.

In the following section we will introduce two classical machine learning models, polynomial regression and the k-nearest neighbours (KNN) algorithm. While regression methods are widely used for time series forecasting, KNN needs to be adjusted to incorporate time series prediction properly. Afterwards we introduce the sub field of deep learning, defining various layers of neural networks and how they are trained efficiently with the backpropagation algorithm.

2.1 Classical machine learning

The mathematical theory in this section is well known and can be looked up in [62, 82].

2.1.1 Polynomial Regression

Regression analysis is a statistical approach used to model and analyse the relationships between a scalar response $y \in \mathbb{R}$ and one or more explanatory variables $x \in \mathbb{R}^n$. When we assume that the relationship of the two is linear, we can use linear regression models. The basic form of simple linear regression refers to the use of a single explanatory variable. If there are more than one explanatory variables, the model is called multiple linear regression. If there are multiple response variables, we call the model multivariate linear regression. In our research regarding regression analysis, we concentrate on detector-wise predictions, hence, do not use multivariate models.

Given the observed value $y \in \mathbb{R}$, explanatory variables $x \in \mathbb{R}^n$, parameters $\beta \in \mathbb{R}^{n+1}$ and the error term $\epsilon \in \mathbb{R}$, the multiple linear regression model is defined as

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \epsilon.$$

In the case of traffic data, the assumption of linearity between response and explanatory variables does not hold. Hence, we use a special case of multiple linear regression called polynomial regression. The underlying model, given m observations

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \ldots + \beta_n x_i^n + \epsilon_i,\ i = 1, 2, \ldots, m$$

can be written as a system of linear equations

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^n \\ 1 & x_2 & x_2^2 & \ldots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \ldots & x_m^n \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_n \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_m \end{bmatrix}.$$

or in matrix notation:

$$y = X\beta + \epsilon.$$

To solve for the parameters, we use the least squares method. The error term $\epsilon \in \mathbb{R}^m$ represents the difference between the observed value and the value predicted by the model. When we are estimating the coefficients using least squares, we are effectively minimizing the sum of the squared errors across all observations. This can be represented by the equation:

$$\min \sum_{i=1}^{m} \epsilon_i^2 = \min \epsilon^T \epsilon = \min (y - X\beta)^T (y - X\beta) := \min S(\beta)$$

To find the β that minimizes $S(\beta)$, we take the derivative of $S(\beta)$ with respect to β, set it to zero and solve for β:

$$\frac{\partial S(\beta)}{\partial \beta} = -2X^T (y - X\beta) := 0$$

This yields the normal equations:

$$X^T X\beta = X^T y$$

From here, we can solve for β:

$$\beta = (X^T X)^{-1} X^T y$$

The inverse $(X^T X)^{-1}$ is guaranteed to exist if $m > n$ and all x_i are distinct since X is a Vandermonde matrix. However, the inverse is computationally expensive to find, hence, in practice we usually use a gradient method to minimize the squared error, e.g. gradient descent.

2.1.2 K-nearest neighbours

The k-nearest neighbours (KNN) algorithm is a non-parametric, lazy learning method used for classification and regression. Given a new observation, it searches the training set for the k training examples that are closest to the observation. Then, it returns the most common output value among them for classification, or an average for regression.

In the context of traffic prediction for a series of timesteps, KNN predicts the outcome variable for a new observation based on the average over each timestep of the outcome variables of its k-nearest neighbours in the training dataset

$$\mathcal{D} = \{(X_t, y_t) | X_t = [x_{t-T}, \dots, x_{t-1}], y_t = [x_t, \dots, x_{t+T'-1}], \forall t = 1, \dots, m \in \mathbb{N}\}.$$

The initiation step in the KNN algorithm involves determining the closest set of neighbours to a given signal X_t based on the distance to potential candidate signals. Hence, we compute the pointwise distance (e.g. MSE) of all signals $X_{t'}, t' \in \{i | i \neq t, i = 1, \dots, m\}$ to the signal X_t and select the k signals with smallest distance.

The prediction $\hat{y}_t$ for a new observation X_t is given by

$$\hat{y}_t = \frac{1}{k} \sum_{i=1}^{k} y_{t_i} = \left[\frac{1}{k} \sum_{i=1}^{k} X_{t_i}, \dots, \frac{1}{k} \sum_{i=1}^{k} X_{t_i+T'-1} \right]$$

where y_{t_i} is the outcome variable for the i-th nearest neighbour to X_t in the training dataset.

We want to note that both KNN and polynomial regression learn models for each sensor in the traffic network independent of each other, there is no transfer of data between sensors occurring in the models. Furthermore if no time features are added, the models do not profit off of the periodic behaviour of the traffic signals like described in Chapter 1. Another way to implement time is by learning multiple functions for each weekday and hour of the day as we do in our works [39, 55, 56].

2.2 Deep learning

As their name suggests, artificial neural networks are inspired by animal brains. Natural neural networks are highly complex information processing systems. Their structure consists of (partly hierarchically) interconnected cores in different areas. The cores themselves each consist of a large number of neurons, which in turn are closely interconnected. Each neuron has a complex morphology and a variety of different non-linear mechanisms used for information processing. Animal brains have a deep architecture. A given input is perceived on several levels of abstraction. Each level of abstraction corresponds to a different area of the cortex. In the first stages, we learn simple concepts and put them together in deeper areas. This can be seen in the human visual system. A signal passes through the ventral visual pathway from the retina to the lateral occipital cortex. The lateral occipital cortex recognizes the signal. The ventral visual pathway consists of a series of areas that process images in an increasingly abstract way, from, e.g., edges, corners, colours and brightness to shapes and patterns to objects. This allows us to learn and recognize three-dimensional objects from seeing two-dimensional images.

Artificial neural networks try to mimic animal brains to learn specific tasks. In this thesis, we address regression tasks, specifically time series forecasting. Artificial neural networks are made up of cores or commonly called layers or cells containing neurons, each of them connected in different ways depending on the type of network. In recent years, neural networks have become increasingly deep, i.e. the number of layers has increased. Most of the time, a neural network learns to solve a task on a specific dataset, e.g. learning to predict traffic flow or speed, solar activity and so on. The network is shown many snippets from these time series present in the dataset. It then produces an output time series predicting the next timesteps. This is called a forward pass or forward step. With the true measurements and a so-called loss function, the error made by the network can be measured. We then use an algorithm, called backpropagation, to tweak the network weights such that next time the network is shown this sample, its output is closer to the true measurements. This is called a backwards pass or backwards step.

In this section, we first introduce a few of the layers that can make up an artificial neural network, namely fully connected layers, graph convolutional layers and dilated causal

convolutional layers. Afterwards, we present the learning process of a neural network. The mathematical theory used in this section is well known and described by Goodfellow, Bengio, and Courville [23] and LeCun, Bengio, and Hinton [41].

2.2.1 Feedforward fully connected neural networks

One of the first types of artificial neural networks are the so-called feedforward fully connected neural networks. These networks consist of neurons or nodes that are grouped into layers. Neurons in the same layer are not connected, but only to neurons in the next layer. In particular, there are no connections back. Information moves only forward, there are no loops. This type of network can be interpreted as a directed acyclic graph (DAG), where the endpoints (neurons) of the edges (connections) are restricted to be located in consecutive layers. An example is shown in Figure 2.1. Feedforward fully connected neural networks are often called dense networks or multilayer perceptrons (MLPs).

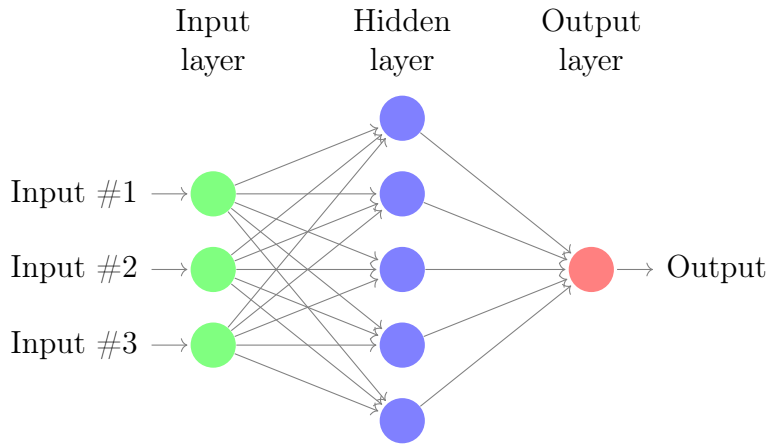


Figure 2.1: A dense neural network with three neurons in the input layer, five neurons in the hidden layer and one output neuron.

The structure of a dense neural network is given by its architecture descriptor $(N_0, N_1, \ldots, N_L)$ where L is the total number of layers (N_0 not included) or the depth of the network and N_ℓ is the number of neurons in the ℓth layer. Each neuron in the ℓth layer is connected to every neuron in the $(\ell-1)$th layer by the relation

$$\sigma(w_n^{(\ell)T} x + b_n^{(\ell)}), \ n = 1, \ldots, N_\ell$$

where $x \in \mathbb{R}^{N_{\ell-1}}$ denotes the input of the neuron, $w_n^{(\ell)} \in \mathbb{R}^{N_{\ell-1}}$ a set of linear weights, $b_n^{(\ell)} \in \mathbb{R}$ a bias and $\sigma : \mathbb{R} \to \mathbb{R}$ the so-called activation function. After computing the outputs of layer ℓ, they are fed to the $(\ell+1)$th layer and so forth for $\ell \in \{1, ..., L\} = [L]$.

The total action of a dense neural network or each layer can be divided into a linear and a non-linear mapping. The linear mapping

$$\begin{aligned} f^{(\ell)} : \mathbb{R}^{N_{\ell-1}} &\to \mathbb{R}^{N_\ell} \\ x &\mapsto f^{(\ell)}(x) = W^{(\ell)}x + b^{(\ell)} \end{aligned} \tag{2.1}$$

with a weighting matrix $W^{(\ell)} = [w_1^{(\ell)}, ..., w_{N_\ell}^{(\ell)}]^T \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ and a bias vector $b^{(\ell)} \in \mathbb{R}^{N_\ell}$, depicts the connection between the neurons of the $(\ell-1)$th and ℓth layer. The non-linear mapping

$$\begin{aligned} \sigma^{(\ell)} : \mathbb{R}^{N_\ell} &\to \mathbb{R}^{N_\ell} \\ x = (x_1, ..., x_{N_\ell}) &\mapsto \sigma^{(\ell)}(x) = \left(\sigma_1^{(\ell)}(x_1), ..., \sigma_{N_\ell}^{(\ell)}(x_{N_\ell})\right) \end{aligned} \tag{2.2}$$

is called the activation function. With the linear mapping (2.1) and the non-linear mapping (2.2), a feedforward network can be constructed as follows.

Definition 2 (Feedforward fully connected neural network)
With (2.1) and (2.2), the function

$$f_{net}(x) = \left(\sigma^{(L)} \circ f^{(L)} \circ \sigma^{(L-1)} \circ ... \circ f^{(2)} \circ \sigma^{(1)} \circ f^{(1)}\right)(x) \tag{2.3}$$

is called a (deep) feedforward fully connected neural network with L layers.

The compositions $\sigma^{(1)} \circ f^{(1)}, ..., \sigma^{(L-1)} \circ f^{(L-1)}$ are called hidden layers and $\sigma^{(L)} \circ f^{(L)}$ the output layer. For $L > 2$, i.e more than one hidden layer, the network is called a deep network. In Figure 2.1 the different layers and the actions of the mappings $\sigma^{(\ell)}$ (circles) and $f^{(\ell)}$ (arrows) are visualized.

While the focus of this thesis lays on convolutional neural networks, fully connected networks or layers are still important, as they often make up the last layers of convolutional neural networks, e.g. to map to the number of outputs. Additionally, convolutional neural networks can be seen as a special case of dense networks.

2.2.2 Convolutional neural networks

Convolutional neural networks or short CNNs are a biologically inspired variation of dense neural networks. Similar to neurons in the visual cortex of animal brains, neurons in CNNs share weights unlike in dense neural networks, where every neuron has its own weight vector associated. Deep dense neural networks can easily use millions of parameters. Weight sharing reduces the number of parameters massively. Additionally, the dimensions of the inputs are kept in contrast to in dense networks, where inputs have to be mapped to a vector. Hence, convolutional neural networks are widely used for many tasks in favour of dense networks, because local neighbourhoods are retained.

As the name suggests, the convolutional neural network is made up of convolutions with a kernel K. These kernels are trained to recognize patterns in the data. By combining convolutions and downsampling operations, the so-called receptive fields of neurons in deep layers grow, allowing them to detect complex patterns. The receptive field in a neural network refers to the part of the image that is visible to a neuron. This receptive field increases linearly as we stack more convolutional layers or increases exponentially combining convolutions and downsamplings.

The kernel dimensions depend on the amount of input features or channels $F \in \mathbb{N}$, output channels $D \in \mathbb{N}$ and chosen kernel size $k \in \mathbb{N}$, i.e. $K \in \mathbb{R}^{D \times F \times k}$. Note, that the kernel dimension does not depend on the length of the input signal, hence, the number of parameters is kept low. The discrete convolution operation in a convolutional layer is defined in the following.

Definition 3 (Convolutional layer) *Let $X \in \mathbb{R}^{F \times N \times T}$ be an input signal to the convolutional layer with $F \in \mathbb{N}$ features, $N \in \mathbb{N}$ nodes and length $T \in \mathbb{N}$. Further, let $D \in \mathbb{N}$ denote the amount of output channels of the layer and $K \in \mathbb{R}^{D \times F \times 1 \times k}$ a kernel with kernel size $k = 2i + 1, i \in \mathbb{N}$. Then, we define the output of the convolutional layer by*

$$X'(p, m, n) = (\bar{X} \boxed{*} K)(p, m, n) = \sum_{q=1}^{F} \sum_{j=-i}^{i} K(p, q, \cdot, j) \bar{X}(q, m, n + j). \tag{2.4}$$

The obtained output feature map $X' \in \mathbb{R}^{D \times N \times T}$ has the same resolution as X, when X is padded in front and after the signal such that $\bar{X} \in \mathbb{R}^{F \times N \times T + 2i}$. Note that the notation assumes that the central component of the kernel is at $K(\cdot, \cdot, 0, 0)$.

2.2.2.1 Dilated causal convolution

Compared to traditional convolutional layers as defined before, dilated causal convolutional layers put an emphasis on preserving the temporal order of data, hence the term causal. They are instrumental when working with sequential data such as time series, where it is essential to maintain causality — meaning the output at any time t should only depend on the inputs at times less than or equal to t.

Incorporating dilation into these convolutions extends the receptive field without increasing the number of parameters or the computational cost appreciably. This is achieved by introducing a dilation rate that skips input values at regular intervals, thereby capturing broader contexts in the input data. The discrete dilated causal convolution operation is defined in the following.

Definition 4 (Dilated causal convolutional layer) *Let $X \in \mathbb{R}^{F \times N \times T}$ be an input signal to the convolutional layer with $F \in \mathbb{N}$ channels, $N \in \mathbb{N}$ nodes and length $T \in \mathbb{N}$. Additionally, let $D \in \mathbb{N}$ denote the amount of output channels of the layer, $K \in \mathbb{R}^{D \times F \times 1 \times k}$ a kernel with kernel size $k = 2i + 1, i \in \mathbb{N}$. At last, let $d \in \mathbb{N}$ the dilation factor and $\bar{X} \in \mathbb{R}^{F \times N \times T + 2di}$ the padded input signal X. Then, we define the output of the dilated causal convolutional layer by*

$$X'(p, m, n) = (\bar{X} \boxed{*} K)(p, m, n) = \sum_{q=1}^{F} \sum_{j=0}^{2i} K(p, q, \cdot, j) \bar{X}(q, m, n - dj).$$

When adopting a dilation rate $d > 1$, the convolution operation skips input values, extending its receptive field without an increase in the computational resources or parameters required. This property facilitates the capture of hierarchical patterns as more extensive contexts are obtained by increasing the dilation rate in subsequent layers.

2.2.2.2 Graph convolution

As the terminology implies, graph convolutional networks (GCNs) implement convolutions on graphs, made up of nodes and edges. Unlike conventional CNNs that operate on a regular grid, GCNs work with irregular data represented as graphs, while seeking to identify and extract local patterns within the neighbourhood of each node.

It is still unclear what the best method is to conduct convolutions on graphs. There are two main research directions. Graph convolution can be defined from the spectral

perspective, where graph filtering and graph wavelets are the main foci [76]. Spectral graph convolutions are defined in the spectral domain based on graph Fourier transform. Doing this allows us to compute the graph convolution by multiplying two Fourier transformed graph signals and then taking the inverse Fourier transform. Another method to define graph convolution is in the spatial domain by aggregating node representations from the node neighbourhoods. In general, graph convolutions are used to aggregate information from neighbouring nodes in a convolutional fashion. Zhang et al. [87] have published a comprehensive review about graph convolutional networks. Zhou et al. [88] additionally cover graph attention and gated graph neural network approaches. In their review, Wu et al. [81] elaborate on spatio-temporal graph convolutional networks as a combination of graph convolution for the spatial domain followed by normal convolution for the temporal domain. This is the approach we are taking in our framework, but the sequence of operations is discovered by a genetic algorithm. Using spatio-temporal graph convolutional networks has been extensively studied for the task of traffic forecasting [17, 61, 79, 84]. All of these methods use a spectral approach. To this end let the graph Fourier transform and its inverse be respectively denoted by $\mathcal{F}$ and $\mathcal{F}^{-1}$. Then the transform of a graph signal X into the spectral domain by $\mathcal{F}$ and its inverse operation are defined as

$$\mathcal{F}(X) = U^T X, \mathcal{F}^{-1}(X) = UX,$$

where U is the matrix of eigenvectors of the normalized graph Laplacian $L = I_N - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$. Here, D is the degree matrix with $D_{ii} = \sum_j (A_{i,j})$ and A the adjacency matrix of the graph. The normalized graph Laplacian has the property of being real symmetric positive semidefinite. Therefore, it can be factorized as $L = U\Lambda U^T$, where Λ is the diagonal matrix of eigenvalues. Then, according to the convolution theorem [54], the convolution operation of the graph signal X with a filter g is defined as

$$g * X = \mathcal{F}^{-1}(\mathcal{F}(X) \odot \mathcal{F}(g)) = U(U^T X \odot U^T g),$$

where $\odot$ denotes the hadamard product and $U^T g$ is the filter in the spectral domain. By simplifying the filter as as a learnable diagonal matrix g_θ, we finally obtain:

$$g_\theta * X = U g_\theta U^T X$$

At this point there is a further split into multiple research directions based on the choice of the learnable diagonal matrix.

Spectral Networks [12, 29] use learnable diagonal matrices $g_\theta = diag(w), w \in \mathbb{R}^N$ as filter. However, this comes with a few disadvantages due to the eigen-decomposition of

the Laplacian. Especially the computational complexity of $O(N^3)$ makes it not viable to use in practice. Defferrard, Bresson, and Vandergheynst [16] propose the ChebNet based on the suggestion, that g_θ can be approximated by a truncated expansion as Chebyshev polynomials $T_k(X)$ [25]:

$$g_\theta \approx \sum_{k=0}^{K} \theta_k T_k(\tilde{L})$$

Here, $\tilde{L} = \frac{2L}{\lambda_{\max}} - I$ is the scaled Laplacian, $\lambda_{\max}$ is the largest eigenvalue of L and I is the identity matrix. T_k can be computed recursively as:

$$T_k(X) = 2xT_{k-1}(X) - T_{k-2}(X)$$

with $T_0(X) = 1$ and $T_1(X) = X$. Consequently, the filter becomes a K-localized operator, meaning it relies only on the K-hop neighbourhood of each node, where K is the order of the polynomial.

One can show via induction that $T_i(\tilde{L}) = UT_i(\tilde{\Lambda})U^T$. Hence, we can simplify the graph convolution to:

$$g_\theta * X = \sum_{k=0}^{K} \theta_k T_k(\tilde{L})X$$

With this, we can define the graph convolutional layer:

Definition 5 (Graph Convolutional Layer) *Let $X \in \mathbb{R}^{T\times N\times F}$ be an input signal to the convolutional layer with $F \in \mathbb{N}$ channels, $N \in \mathbb{N}$ nodes and length $T \in \mathbb{N}$. Moreover, let $D \in \mathbb{N}$ denote the number of output channels and $K \in \mathbb{R}^{D\times F\times K}$ be a tensor of Chebyshev coefficients where K is the order of the polynomial. Then, we define the output of the graph convolutional layer by*

$$X'(p, m, n) = \sum_{q=1}^{F} \sum_{k=0}^{K} K(p, q, k)T_k(\tilde{L})X(\cdot, \cdot, q),$$

where $\tilde{L} = 2L/\lambda_{max} - I_N$ is the normalized Laplacian with λ_{max} being the largest eigenvalue of L and I_N is the $N \times N$ identity matrix. The Chebyshev polynomials $T_k(\tilde{L})$ are

applied to this rescaled Laplacian, yielding a matrix for each order k. These matrices are applied to each channel of the input feature matrix X.

For each node i in the graph, the output feature value $X'(p, m, n)$ depends on a weighted sum of the input features of all nodes, where the weights are determined by the Chebyshev coefficients and the graph structure encoded in the Laplacian L.

In practice, the order K of the Chebyshev polynomial is often chosen to be relatively small to reduce the computational complexity and to prevent overfitting. For example, Kipf and Welling [35] simplify the layer by setting $K = 1$ and $\lambda_{\max} = 2$.

Deep graph convolutional neural networks can suffer from performance degradation due to oversmoothing, overfitting and training instability [42]. Zhou et al. [89] show that normalizing each node during training can greatly decrease these problems. Hence, in our framework, we apply their node normalization after each graph convolutional layer. In our experiments, this lead to great improvements compared to other techniques such as batch normalization [32], where normalization is done for each mini-batch of data.

2.2.3 Backpropagation

We have presented commonly used layers in convolutional neural networks. In the following, we describe the basics of the optimization (also learning or training) process in which the weights of a neural network are computed.

Similarly to fully connected networks, convolutional networks can also be described as a mapping

$$f_\Theta(X) = \left(\sigma^{(L)} \circ f^{(L)} \circ \sigma^{(L-1)} \circ ... \circ f^{(2)} \circ \sigma^{(1)} \circ f^{(1)}\right)(X) \in \mathbb{R}^{N \times T},$$

where $f^{(\ell)}$ are layers as presented in the previous section, $\sigma^{(\ell)}$ are activation functions, Θ the network parameters and $N \times T$ is the dimension of the output. Since we especially consider two-dimensional data in this thesis (although the concepts can easily be applied for one- and three-dimensional data), the inputs to the network are graph signals $N \times T$ and are made up of F channels, e.g. flow, speed and other features. Commonly, inputs to the network are not made up of one sample, but rather, batches made up of multiple random samples are propagated through the network at once. The amount of samples in a batch B is called the batch size $n_B \in \mathbb{N}$. For simplicity, we explain the concepts of learning for a batch size of $n_B = 1$ in the following.

As before, a data set consists of pairs $\{(X_t, y_t)\}_{t=1,\dots,m\in\mathbb{N}}$, where $X_t \in \mathbb{R}^{F\times N\times T}$ is the input samples and $y_t \in \mathbb{R}^{D\times N\times T}$ ground truth labels (measurements). Given an input X_t, the network predicts

$$\hat{y}_t = f_\Theta(X_t).$$

Especially in the beginning, when network parameters are mostly random, the predicted labels $\hat{y}_t$ are dissimilar from the ground-truth class labels y_t. We quantify the dissimilarity with a loss function as mentioned in the beginning of this chapter. For all the networks in this thesis, we use the mean squared error (MSE), which measures the euclidean distance between the time series. It is defined as

$$MSE(y_t, \hat{y}_t) = \frac{1}{m}\sum_{t=1}^{m}(y_t - \hat{y}_t)^2$$

The training goal is to find network parameters Θ^* such that

$$\Theta^* = \underset{\Theta}{\operatorname{argmin}} \sum_{t=1}^{m} MSE(y_t, f_\Theta(X_t)).$$

Backpropagation is an efficient way to compute the gradients of the network, which are necessary for most optimization algorithms. The gradient of the loss with respect to every single weight has to be computed. To do so, we use the chain rule of differentiation. We do not go through the whole backpropagation algorithm for convolutional neural networks, as the formulas can get quite large, but we show the key concepts. We compute the gradient of each parameter in every layer of the network. For example, let $k_\iota^{(\ell)}$ denote the kernel weight in the ℓth layer at position $\iota \in \{-i_1, \dots, i_1\} \times 1$. For simplicity, the amount of channels is always 1. We compute the gradient of the MSE with respect to $k_\iota^{(\ell)}$, i.e.

$$\frac{\partial MSE}{\partial k_\iota^{(\ell)}}. \tag{2.5}$$

When $f^{(\ell)}$ is a convolutional layer, let

$$x_{m,n}^{(\ell)} = f^{(\ell)}(o^{(\ell-1)}) = \sum_{\iota=-i_1}^{i_1} k_\iota^{(\ell)} o_{m+\iota,n}^{(\ell-1)}$$

be the (m,n)th value of the output of the convolution operation in layer ℓ as defined in (2.4), where $o^{(\ell)} = \sigma^{(\ell)}(x^{(\ell)})$ denotes the activations of the ℓth layer. Using the chain rule of differentiation for the MSE gradient (2.5), we obtain

$$\frac{\partial MSE}{\partial k_\iota^{(\ell)}} = \sum_{n=1}^{N}\sum_{m=1}^{T} \frac{\partial MSE}{\partial x_{m,n}^{(\ell)}} \frac{\partial x_{m,n}^{(\ell)}}{\partial k_\iota^{(\ell)}},$$

where N and T are the amount of nodes and length of the output feature map. The derivative of the output value with respect to the kernel weight can easily be computed, hence, we obtain

$$\frac{\partial MSE}{\partial k_\iota^{(\ell)}} = \sum_{n=1}^{N}\sum_{m=1}^{T} \frac{\partial MSE}{\partial x_{m,n}^{(\ell)}} o_{m+\iota,n}^{(\ell-1)} = \sum_{n=1}^{N}\sum_{m=1}^{T} \delta_{m,n}^{(\ell)} o_{m+\iota,n}^{(\ell-1)}.$$

This means that the gradient of the MSE with respect to a kernel weight depends on the previous layers output and the gradient of the MSE with respect to the output value of the layer the kernel weight belongs to. We can further compute the $\delta_{m,n}^{(\ell)}$ using the chain rule

$$\delta_{m,n}^{(\ell)} = \frac{\partial MSE}{\partial x_{m,n}^{(\ell)}} = \sum_{Q} \frac{\partial MSE}{\partial f^{(\ell+1)}(o^{(\ell)})_Q} \frac{\partial f^{(\ell+1)}(o^{(\ell)})_Q}{\partial x_{m,n}^{(\ell)}}$$

Here, the set $Q \in \mathbb{N}^2$ denotes the index set of values in the output $f^{(\ell+1)}(o^{(\ell)})$ that are affected by $x_{m,n}^{(\ell)}$. We can not specify Q, because following the convolutional layer might be any layer, e.g. another convolution or skip connection. Hence, the affected values might differ. Note that

$$\delta_{m,n}^{(\ell)} = \sum_{Q} \delta_{Q}^{(\ell+1)} \frac{\partial f^{(\ell+1)}(o^{(\ell)})_Q}{\partial x_{m,n}^{(\ell)}}.$$

Therefore, $\delta^{(\ell)}$ can be computed recursively while the second gradient is known beforehand (dependent on the used layer and activation). The recursive computation of $\delta^{(\ell)}$ starts from the last layer and propagates backwards through the network to the first layer giving this process its name backpropagation.

After computing the gradients via backpropagation, we can update the network parameters with gradient descent

$$\Theta :- \Theta - \eta \frac{\partial MSE}{\partial \Theta},$$

where $\eta \in \mathbb{R}^+$ is called the learning rate. When all samples in the training set (a subset of the whole dataset) are propagated through the network once and the parameters updated, we say that an epoch has passed. The samples are usually shuffled before starting the next epoch and some kind of data augmentation might be applied to combat overfitting.

3 Neural architecture search

With advances in computational hardware, it is more feasible to train large neural networks today than ever before. It is not unusual for a network to consist of hundreds of layers made up of billions of parameters. Most of these parameters are trainable parameters, e.g. (kernel) weights and biases. The number of hyperparameters grows with ever increasing network sizes. We are faced with lots of choices:

- **Depth of the network**
  One has to decide how many layers are used in a neural network. More layers can help in finding and combining patterns within the data. However, with a growing number of layers, the number of parameters usually grows. This can increase the risk of overfitting to the training dataset, decreasing the performance on the test dataset.
- **Layer operation and connections**
  The choice of operation at each depth is also crucial. One has to choose which operations to apply and also in which order. In ResNet presented by He et al. [27] it was found that adding skip connections after convolutional blocks aids in training the networks and leads to better performance. However, different orders of operations within the ResNet blocks lead to substantially different results [28].
- **Kernel size of convolutions and K-hop neighbourhoods of graph convolutions**
  The kernel size impacts the receptive field of a neural network, i.e. how many surrounding pixels or values can be seen. There is no best choice of kernel size or K-hop neighbour for nodes. Hence, these hyperparameters have to be chosen for the dataset at hand.

When tuning a neural network, it feels like this list goes on forever. Some of these choices can be made based on self-made experience or experience from other researchers. A common approach is a grid search over many possible hyperparameters, i.e. for each combination of hyperparameters, we train a neural network and evaluate the performance on a held-out validation set. We can then choose the hyperparameter configuration with the best result on the validation set. Grid search, however, suffers from the so-called curse of dimensionality [8, 9] as it can easily explode due to the sheer amount of possible hyperparameter combinations growing exponentially. In recent years a new research branch called neural architecture search (NAS) directed at these problems has emerged

[2, 5, 10, 22, 24, 47, 65, 67, 77, 90]. While being more efficient, earlier improvements of grid search, e.g. random search [11] for hyperparameter optimization do not optimize the architecture directly. Neural architecture search has evolved to automatically discover novel neural network architectures. In this chapter we give an overview of the history of neural architecture search before diving deeper into the main approach used in this thesis: evolutionary neural architecture search.

3.1 Literature review

One of the earliest NAS methods is called neuro-evolution [22, 24, 77]. Evolutionary algorithms are inspired by the evolution in nature. Some candidate solutions to a specific problem are (randomly) initialised, e.g. in our case simple neural networks and their so-called fitness evaluated. The fitness measures the performance of a candidate on the task. Based on the fitness of each candidate solution, they are then mutated in some way depending on the framework.

Early neuro-evolution approaches specify a fixed fully connected neural network topology or architecture and evolve the weights. This is contrary to most neural network optimization schemes, which use backpropagation. It is debatable whether only optimizing the weights is sufficient to find the best approximation. Gruau, Whitley, and Pyeatt [24] compare evolving of the weights of a fixed topology and evolving the weights and the topology simultaneously. They conclude that although evolving the architecture additional to the weights takes more time for the algorithm, the architectures found are smaller and a better fit for the specific problem. In contrast, it takes immense human effort, experience and time to design the fixed architecture. Their idea, however, was later compromised by Gomez and Miikkulainen [22], who have shown that randomly choosing the number of hidden neurons is a superior method. Stanley and Miikkulainen [77] presented the NEAT method that increases the efficiency of weight and topology optimization, leading to superior results on benchmark learning tasks over methods with fixed topology, including the method by Gomez and Miikkulainen [22]. The basic idea is to alternate the optimization of weights and architecture in order to balance the fitness and diversity of candidates.

When there is a lot of data at hand, we require large neural networks. However, the above approaches create too small networks, because their goal is to be efficient instead of act on a large scale. Real et al. [69] search for large models on the CIFAR-10 and CIFAR-100 datasets (image classification). Instead of allowing mutations such as adding a single node to the network, their algorithm can mutate whole layers, adding hundreds of nodes at a time, thereby increasing expressive power quickly. This extensive search is slow, but the discovered architectures can reach competitive accuracies in comparison to hand crafted designs.

Search based methods like evolution are generally slow and computationally expensive because they require a massive amount of sampled candidates, each of which has to be trained until convergence to measure its fitness. Hence, approaches moved away from evolution and focus on reinforcement learning (RL). Zoph and Le [90] use a recurrent neural network (RNN) to predict hyperparameters of a convolutional neural network trained with a policy gradient method. They remark that the structure and connectivity of a neural network can generally be specified by a string of variable length. This makes it possible to sample topologies using RNNs. In their method, they use the RNN to predict one hyperparameter of a network at each timestep until they obtain a complete architecture. The resulting network is trained until it converges. Then, the accuracy on a validation set is computed. RL algorithms try to maximize the reward given a problem. The reward signal can be based on the performance of the networks as in the case of Zoph and Le [90], but it can also include other criteria.

The idea of neural architecture search is also used in following works. Since the method proposed by Zoph and Le [90] is still slow when used on large datasets, Zoph et al. [91] propose to learn a single cell which can be stacked in series to get an arbitrary model size instead of learning the whole network. Cells or layers are made up of nodes, each connected by different commonly used operations. These operations make up the so-called NASNet search space, e.g. identity, convolutions, dilated convolutions, separable convolutions with different kernel sizes, max-pooling with different sizes. The RNN can at each time step decide which of these operations get applied to the preceding hidden state. The resulting architecture is a multi-layer network made up of the learned cell, each with the same structure but with different weights. Their method, however, still requires learning the sampled architecture from scratch to obtain the reward signal. The search takes between 32,000 and 43,000 GPU hours. GPU hours are defined as hours trained on a GPU. As large models are often trained in parallel on multiple GPUs, it is necessary to report the total amount of hours spent over all these GPUs for fair comparison. Additionally, it has to be stated which GPU is used, since there can be great performance differences. Due to this, researchers need to train all models they want to compare on the same GPUs for a fair comparison and can only cite results when the same GPUs were used in the original work. Pham et al. [65] observe that all the iterations in NAS can be seen as iterating over sub-graphs of a larger super-graph. Hence, it is possible to represent the search space of NAS by a single Directed Acyclic Graph (DAG). With this method, weight sharing of all sub-graphs can be made possible. The sampled architectures do not have to be trained from scratch each time, but can share weights. An architecture can then be sampled as a sub-graph with policy gradient. The only differences between the architectures are the paths chosen in the DAG. This leads to a speed-up of 1000 times compared to the previous method, hence, Pham et al. [65] call their method Efficient NAS. However, the speed-up brings with it slightly worse results on CIFAR-10 than the original approach by Zoph et al. [91] (2.89% compared to 2.65% error), despite using 30% more parameters. Ashok et al. [2] use RL NAS to compress big neural networks into smaller ones to make them applicable in the real world. Their algorithm takes the larger teacher model as input and outputs the

smaller student network. The reward used for the RL algorithm is computed from the students evaluation performance and compression. They can compress neural networks by one magnitude while maintaining similar performance to the teacher. However, using this method somewhat defeats the purpose of NAS, since the teacher networks are hand crafted and have to be selected for the algorithm.

It has become apparent, that all of the presented approaches come with scalability issues or when sped up do not perform sufficiently. Liu, Simonyan, and Yang [47] address these problems by relaxing the search space such that it becomes continuous instead of discrete. They do this by applying all possible operations at each edge of the DAG and learn a weighting of the outputs (mixed operation). Their approach, called DARTS, makes it possible to optimize the architecture (weights) by gradient descent with respect to the validation loss. They use a super-network made up of cells with the same topology. This method was introduced by Zoph et al. [91]. While the cells share their architecture, the (e.g. kernel) weights can be learned individually. The architecture and the weights are trained alternately on the validation set (for the architecture) and the training set (for the weights) with gradient descent. This is done to obtain an optimal network architecture and associated weights for the given data set. After the search, they obtain a promising cell and stack it multiple times to build a new neural network. This (final) network is then trained from scratch. While DARTS improves run time tremendously, the search process is biased as a result of the bi-level optimization and greedy behaviour of gradient descent. Their algorithm tends to prefer skip connections [60] and its performance collapses after a large number of epochs [44]. Mun, Ha, and Lee [60] solve this issue by introducing dynamic exploration in DE-DARTS. Liang et al. [44] introduce a simple early stopping in their DARTS+ to tackle the overfitting problem. After the search process, a discrete architecture has to be derived, since applying mixed operations is computationally expensive. When the weights of the mixed operations converge against a one hot encoding, the discovered and final architectures are the same. In practice, this is not always the case leading to big differences in learned and discrete architecture. The transfer from the smaller network with less cells to the larger network with more cells often creates a gap in performance as found by Chen et al. [14]. To alleviate this problem, they use an algorithm that allows the network depth to grow slowly. However, this results in higher computational overhead and search instability necessitating search space approximation and regularization.

Very recently, there have been advances in performance estimation techniques for neural architectures [7, 52, 57, 74]. Performance estimation refers to the process where a neural network is not trained until convergence to obtain the performance, but it is estimated. This estimator can be a shortened learning process, e.g. training fewer epochs or batches per epoch and using the best performance during shortened training as an estimator, or doing additional extrapolation of the learning curve [18]. Similarly, training speed estimators estimate how fast a neural network can learn [72]. This is usually done by training a few epochs and computing a weighted sum over the losses of each epoch. Model based estimators learn a prediction function from architecture

feature encodings. Hybrid methods combine extrapolation and model based estimators. There are also the previously mentioned weight sharing techniques in a super-network as used in e.g. DARTS. Most recently, zero-cost (ZC) proxies, originating from pruning of architectures, have been explored as performance estimators [1, 49, 50, 57, 80]. Pruning for neural networks refers to the process of removing certain elements of a network, such as neurons, weights, or even entire layers, to reduce the model's size and complexity while maintaining (or even improving) its performance. The goal of pruning is to achieve a more efficient neural network that can be deployed in resource-constrained environments like mobile devices or edge devices without a significant loss in accuracy. To do so, pruning methods use measures for each part of the neural networks. ZC proxies use the same measures but applied to the whole architecture. This results in a score that is used to rank neural architectures. Evaluating ZC proxies usually involves passing one batch through the neural network and applying the metric to the activations or outputs. For some ZC proxies a backwards pass to compute gradients is necessary. It can easily be seen that these methods are much faster in estimating performance of neural networks than training until convergence or early stopping, but might come at the expense of correlation between true performance and the score.

To conclude, the objective of NAS is to find an optimal architecture A from the space of architectures $\mathcal{A}$ that minimizes the loss function $\mathcal{L}$ on a given dataset $\mathcal{D}$. To be more precise, we want to solve a bi-level optimization problem:

$$A^* = \min_{A \in \mathcal{A}} \mathcal{L}(\theta^*(A), A, \mathcal{D}_{\text{valid}}) \tag{3.1}$$

$$\text{s.t.} \quad \theta^*(A) = \arg\min_{\theta} \mathcal{L}(\theta, A, \mathcal{D}_{\text{train}}) \tag{3.2}$$

Here, $\mathcal{D}_{\text{train}} \subset \mathcal{D}$ and $\mathcal{D}_{\text{valid}} \subset \mathcal{D}$ respectively denote the training and validation datasets and θ the network parameters.

To solve the bi-level optimization problem, NAS can be split into three components: the architecture search space (definition of $\mathcal{A}$), the search method (upper level objective function) and the performance estimation (lower level objective function). In this thesis, we want to explore the application of evolutionary NAS to traffic forecasting and if the use of ZC proxies makes this method viable to use in applications. To our knowledge, the only previous research on NAS for our traffic prediction problem has been conducted by Rahimipour, Moienfar, and Hashemi [66], Pan et al. [64], Ke et al. [33] and Chen et al. [13]. Rahimipour, Moienfar, and Hashemi [66] use an evolutionary algorithm to optimize the number of neurons in a two layer fully connected neural network and the ratio and slope of the activation function on a small dataset with only three sensor locations. Their method produces satisfactory results. Pan et al. [64] and Ke et al. [33] use differentiable NAS and call their frameworks AutoSTG and AutoSTG+. In addition they employ meta learning to learn adjacency matrices for spatial graph convolution and kernels for

temporal convolution from meta knowledge of the attributed graph. Chen et al. [13] build on the DARTS framework adding a multi-scale decomposition transforming raw time series to multi-scale sub-series prior to training. The adaptive graph learner finds inter-variable dependencies for the different time scales, i.e. they learn a general adjacency matrix and scale-dependent ones. Our framework is the first one to employ evolutionary NAS with convolutional operations for the task of traffic forecasting. Furthermore, it is the first to apply zero-cost proxies to the traffic prediction task.

In the following, we will present our framework in detail. The search space, i.e. the set of all possible neural networks that is described in Section 3.2, the search method in Section 3.3 and the performance estimation in Section 3.4.

3.2 Search space

The search space of NAS consists of all possible neural networks that can be formed by selecting size, connections and operations. It includes the operations and choices of their hyperparameters and the structure of the neural network. A neural network is usually learned fully or split into cells with the same structure stacked on top of each other to obtain a larger network [45, 46, 68, 91]. Searching for a full architecture can be seen as a special case of the cell based approach where just one cell is used for the final architecture.

Each cell consists of nodes $x^{(i)}, i \in \{-2, -1, 0, \dots, N-1 | N \in \mathbb{N}\}$. The nodes $x^{(-2)}$ and $x^{(-1)}$ are the inputs of the cell. The other nodes can be seen as feature maps. The nodes are ordered in a sequence, forming a Directed Acyclic Graph (DAG). Each edge (i, j) is associated with an operation $o^{(i,j)}$ mapping the node $x^{(i)}$ to a feature map. To obtain the node $x^{(j)}$ all of its preceding nodes are summed up:

$$x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)}), j \in \{0, \dots, N-1\}$$

Each cell has as input nodes $x^{(-2)}$ and $x^{(-1)}$ the outputs of the two preceding cells. For the first cell in the network, $x^{(-2)}$ and $x^{(-1)}$ are both the input graph signal. The output of the cell is the sum or concatenation of the nodes $x^{(j)}$ for $j \in \{N-M, \dots, N-1\}$. The parameter $M \in \mathbb{N}$ controls how many nodes are concatenated. The amount of features or channel depth usually stays the same in a cell. The cell based approach has the disadvantage that it adds two hyperparameters one has to search over, namely the amount of cells in the search architecture and the amount of nodes in the cell. Hence, it has to be run multiple times for those.

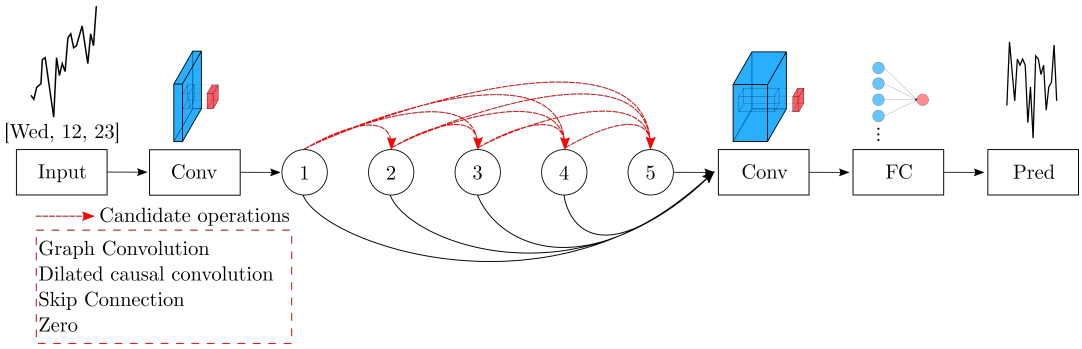


Figure 3.1: Architecture search space of our framework.

In our framework, we learn the whole architecture, eliminating the need to search over the amount of cells in the search architecture. The general structure of the resulting neural networks can be seen in Figure 3.1. The inputs to the networks are the last T traffic signals $X \in \mathbb{R}^{F \times N \times T}$ as described in the traffic prediction problem 1 in Chapter 1 and the convolutional layer definitions in Chapter 2 with some feature vector incorporating time information, e.g. weekday, hour, minute of each timestamp. We apply a 2 dimensional convolutional layer with 1×1 kernel before applying the first operation in the DAG as found by the search algorithm. The candidate operations that can be chosen at each edge depend on the task. For image classification usually normal, dilated and separable convolutions are used [47]. For the task of traffic predictions, we use the operations described in Section 2.2, i.e. graph convolution and dilated causal convolution. Additionally, the operation space often includes skip connections, which apply a convolution with 1×1 kernel to the input, and none, which returns just zeros. After the DAG, we apply another 2 dimensional convolution with 1×1 kernel size and a feedforward fully connected neural network for downsampling the channel depths and scaling. We obtain the output graph signal $X \in \mathbb{R}^{D \times N \times T'}$ with T' timestamps into the future and D features, where $D = 1$, since we only predict flow or speed.

Note that our networks do not have a boundary on the size, i.e. they can become as large as the search method allows. Since we learn the full architecture in contrast to a cell, it is sufficient to use one input node (the input graph signal). The channel depth increases by a factor of two throughout the network until a maximum possible depth is reached. Hence, the channel depth $n_c(i)$ of each node $x^{(i)}$ is defined by

$$n_c(i) = \min(2^i n_c(0), n_{c,max}), i \in \{1, \dots, N-1\},$$

where $n_c(0), n_{c,max}$, respectively, are the starting and maximum channel depth that can be chosen to control the amount of network parameters.

3.3 Evolutionary algorithms

NAS requires navigating through vast and complex search spaces, which is challenging due to the high dimensionality and discrete nature of the problem. To address these challenges, meta-heuristic algorithms have emerged as powerful tools. Among these, three significant evolutionary algorithms, namely Genetic Algorithm (GA), Differential Evolution (DE) and Particle Swarm Optimization (PSO), are analysed in the following to determine the most suitable approach for our task.

Genetic Algorithm (GA) is an evolutionary algorithm inspired by the mechanisms of natural selection and genetics [21, 30]. It operates on a population of candidate solutions, simulating natural evolutionary processes such as selection, crossover and mutation. Exploitation is achieved through the selection process, focusing on individuals of higher fitness to exploit the fitness information within the population [19]. In Elitism selection, the candidates with highest fitness are picked, while in binary tournament selection two candidates are drawn randomly from the population and the one with higher fitness is selected. Crossover mixes traits from paired parent solutions to create offspring, introducing new solution variations. Mutation randomly alters parts of a solution. The genetic operators (crossover and mutation) facilitate exploration by creating diversity and introducing new genetic structures into the population [19]. GA's efficacy in NAS stems from its robustness in handling discrete and unordered search spaces and its ability to explore a diverse range of solutions. Additionally, it is highly parallelizable since candidate solutions are not dependent on each other.

Differential Evolution (DE) is a population-based optimization method known for its success in continuous optimization problems [78]. It distinguishes itself with a unique mutation strategy where differences between randomly selected individuals are used to perturb other individuals in the population. DE was designed for continuous spaces, hence, its mutation strategy might not be as effective for the discrete, categorical choices characteristic of NAS search spaces. Awad, Mallik, and Hutter [3] transfer the discrete operation space into a continuous one by defining a sub interval in $[0, 1] \in \mathbb{R}$ for each operation. This makes it viable to use the algorithm and leads to superior performance compared to other search algorithms like random search and regularized evolution. DE supports parallel execution, as each individual in the population can be evaluated independently.

Particle Swarm Optimization (PSO) inspired by the collective behaviour observed in natural swarms like birds and fish, employs a set of particles that move through the search space [34]. Each particle's movement is influenced by its own best-known position and the best-known positions of its neighbours. This social sharing of information guides the swarm towards optimal regions in the search space. PSO, originally designed for continuous spaces, has been adapted for discrete search spaces in NAS [15, 48]. This adaptation often involves quantizing the search space or using binary versions of PSO.

Exploration and exploitation are balanced through particle movement. Particles explore the search space by following their own and their neighbours' best-found positions, allowing a mix of personal and social learning. It is inherently parallel, as particles can be updated simultaneously.

In our work we selected GA as the algorithm to use in our search method, because there is no need to transform the search space like in DE and PSO. Additionally, the genetic operators mutation and crossover maintain a diverse population. This is also the case in DE. In PSO, the particles tend to converge to local optima due to the nature of the particle movement towards the best known positions. Due to time restrictions, in this work we did not have the chance to run a comparative simulation among GA, DE and PSO. However, in future work we plan to tackle this point in a comprehensive way.

The GA utilized in this study follows a structured approach, encompassing several stages. At the initialization we generate a diverse population of neural network architectures, with each individual representing a unique architecture. This diversity is crucial for a comprehensive exploration of the search space. Then, we iterate through the genetic operations and selection process. The selection process prioritizes architectures based on their fitness, favouring those more suited to the task. Crossover and mutation introduce new architectures into the population, ensuring continuous exploration and adaptation. The algorithm terminates after a predetermined number of generations or when a certain performance threshold is achieved. This termination criterion is essential to ensure that the search process is both efficient and effective. The pseudo code of our genetic algorithm is shown in Algorithm 1 and described in more detail in the following sections.

3.3.1 Initialization

The algorithm begins by creating an initial comparatively large random population of size $n_w \in \{2n|n \in \mathbb{N}\}$ as a so called warm-up population. Each architecture within the search space has an equal probability of selection. To obtain their fitness we use a performance estimator as described in Section 3.4. From this warm-up population, we choose the top $n_p \in \{2n|n \in \mathbb{N}\}$ architectures for the following cycles. This method of selection is called elitism.

3.3.2 Evolutionary operators

Once the population is initialized and evaluated, the core iterative process starts. This involves repeatedly performing crossover and mutation for $n_c \in \mathbb{N}$ generations. To apply crossover, we need to select two parents from the current generation. We use the binary tournament method to select parent solutions for crossover. In binary tournament

Algorithm 1 Genetic algorithm for neural architecture search.

Require: $n_w > 0, n_p > 0, n_c > 0$

$population \leftarrow \varnothing$
$best \leftarrow \varnothing$
while $|population| < n_w$ **do**
    $model.arch \leftarrow$ RandomInit()
    $model.fitness \leftarrow$ PerformanceEstimation($model.architecture$)
    add *model* to *population*
end while
$population \leftarrow$ Elitism($population$)
for i in $range(n_c)$ **do**
    $offspring \leftarrow \varnothing$
    while $| offspring | < n_p$ **do**
        $parents \leftarrow$ BinaryTournament($population, 2$)
        $children \leftarrow$ UniformCrossover($parents$)
        add *children* to $offspring$
    end while
    for *model* in $offspring$ **do**
        $model.arch \leftarrow$ Mutate($model.architecture$)
        $model.fitness \leftarrow$ PerformanceEstimation($model.architecture$)
        if $model.fitness < best.fitness$ **then**
            $best \leftarrow model$
        end if
    end for
    add $offspring$ to *population*
    $population \leftarrow$ BinaryTournament($population, n_p$)
end for
return *best*

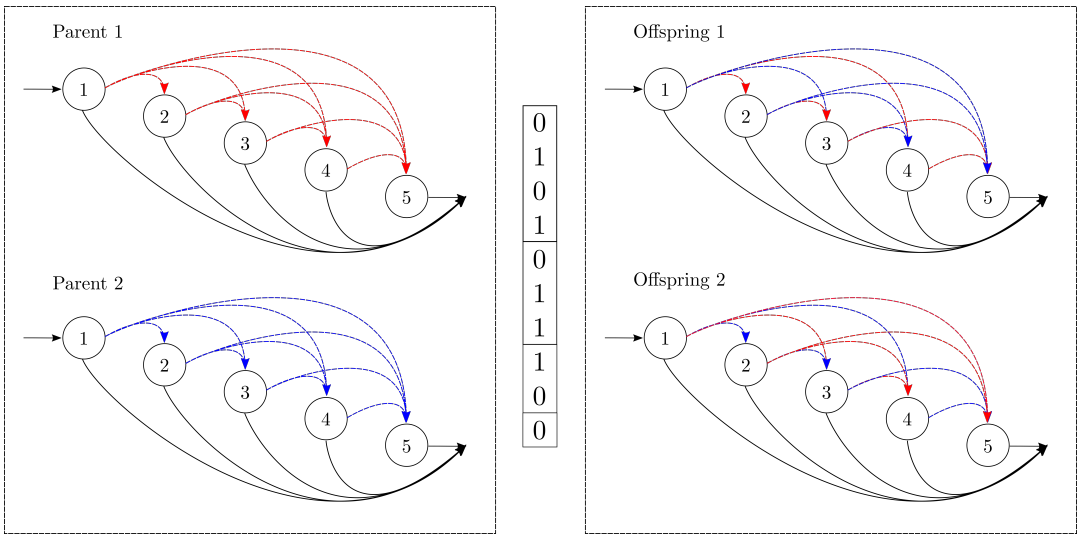


Figure 3.2: Uniform crossover generates offspring by uniformly sampling and switching edges from parents. The binary vector in the middle is an uniformly sampled encoding of the switched edges. If there is a 0, the first child gets the edge from the first parent and the second child from the second parent. If there is a 1, the first child gets the edge from the second parent and the second child from the first parent.

selection, two solutions (or chromosomes) are randomly chosen. The one with superior fitness gets selected. This method ensures that well-performing architectures have a higher likelihood of being chosen for reproduction while still preserving diversity [58].

Crossover combines genetic material from two parent architectures. We employ uniform crossover, which involves selecting a random subset of nodes from the parents' architectures to swap. If there is a discrepancy in the sizes of the two architectures, the operation is performed up to the size of the smaller parent, ensuring the resulting children retain the original sizes of the parents. This process is shown in Figure 3.2. Other methods for crossover include one-point crossover and k-point crossover. One point or node is chosen randomly and the genetic material of each parent on the right of that point is swapped, resulting in two offspring. If there are k points, the genetic material up until the second point is swapped and then again from the third point and so forth.

Subsequent to crossover, mutation operations are applied, introducing slight variations to promote diversity and explore new regions of the solution space. The potential mutation operations are outlined in the following Sections.

3.3.2.1 Switching edges

Switching edges involves the swapping of operations between two distinct edges. Consider two edges (i, j) and (i', j') with associated operations $o^{(i,j)}$ and $o^{(i',j')}$, respectively. The mutation exchanges the operations, thereby rerouteing the connectivity within the DAG. This mutation allows the GA to probe architectures with similar components but differing connectivity patterns.

3.3.2.2 Removal of an operation

For an edge (i, j) with an associated operation $o^{(i,j)}$, the removal mutation sets this operation to zero, effectively removing the connection between nodes i and j. In the DAG framework, this means the directed link between these nodes is eliminated. Such a mutation can lead to reduced complexity, potentially streamlining the architecture and reducing overfitting.

3.3.2.3 Altering the type of operation

This mutation operation alters the type of operation on a specific edge. For an edge (i, j) with its associated operation $o^{(i,j)}$, the mutation swaps this with another operation o' chosen randomly from the operation set $\mathcal{O}$.

3.3.2.4 Modifying parameters of an operation

This mutation is targeting the hyperparameters within the architecture, such as kernel size or dilation factors. For instance, for a convolutional operation on edge (i, j) denoted as $o^{(i,j)}$, this mutation might tweak the kernel size. Such modifications offer a fine-grained exploration mechanism within the architectural domain.

3.3.2.5 Node addition or removal

One of the more transformative mutations, this operation either introduces or eliminates nodes. Adding a node implies introducing a new operation from the set $\mathcal{O}$ (excluding none) connecting the new node to existing ones. In contrast, node removal entails deleting the node and all its associated operations. This mutation allows the GA to explore different architecture sizes. Theoretically, architectures can become infinitely large, however, due to overfitting, they would not score well and, therefore, are less likely to be selected.

Crossover and mutation are not applied to every pair of parents and not to every child. Given the mutation and crossover methods are chosen, there are two additional hyperparameters, namely crossover probability $P_c \in [0, 1]$ and mutation probability $P_m \in [0, 1]$ that control the evolution process. A high P_c ensures a rapid mixing of genetic material, promoting diverse solutions, while a low P_c maintains more of the original solutions, favouring exploitation over exploration [19]. The choice of P_c largely depends on the problem domain and characteristics of the search space [26]. It is essential to fine-tune P_c using cross-validation or other validation techniques specific to the problem at hand. The mutation probability, P_m, determines the chance that a particular gene (or component) of a solution will undergo mutation. This probability is typically set much lower than P_c since mutation acts as a background operator to introduce randomness and maintain diversity. The value should be chosen considering the size of the genome (i.e., the number of components in a solution, e.g. number of layers and hyperparameters of those). For instance, if a solution has 100 components and $P_m = 0.01$, on average, one component will be mutated when mutation is applied to a solution. As with P_c, the choice of P_m should be based on empirical tests, cross-validation, or domain knowledge about the specific problem.

Following crossover and mutation, each resulting child's performance is estimated. Then, we use binary tournament selection to select the final population of the current cycle.

3.3.3 Termination

After a set number of evolution cycles n_c or if the fitness does not improve any more for some cycles, the algorithm concludes, returning the solution (i.e., neural network architecture) with the best observed fitness across all generations. Then, a hyperparameter tuning process succeeds GA, where the best hyperparameters, e.g. batch size and channel depths as described in Section 3.2 are found. The result is an architecture that should perform superior to most other architectures in the search space on the given test dataset.

3.4 Performance estimation

As discussed in Section 3.1, the biggest computational bottleneck in NAS is the performance evaluation of each neural network architecture. In the upper level of the bi-level optimization problem (3.1)

$$A^* = \min_{A \in \mathcal{A}} \mathcal{L}(\theta^*(A), A, \mathcal{D}_{\text{valid}})$$

an algorithm tries to find the optimal architecture $A^* \in \mathcal{A}$ on the validation dataset $\mathcal{D}_{\text{valid}}$ given optimal parameters $\theta^*(A)$ of each architecture $A \in \mathcal{A}$ on the training data $\mathcal{D}_{\text{train}}$ as described in the lower level (3.2):

$$\theta^*(A) = \arg\min_{\theta} \mathcal{L}(\theta, A, \mathcal{D}_{\text{train}})$$

Finding the optimal parameters $\theta^*(A)$ for each architecture involves training until convergence, sometimes for multiple hyperparameter combinations of each architecture in the search space. Since the search space can be made up of thousands of architectures (or technically be infinitely large), this takes an extensive amount of time. Hence, we generally want to use some performance estimation technique to approximate the lower level.

Early NAS methods [68, 90] suffer tremendously w.r.t. computation time, taking thousands of GPU hours per run. One GPU hour refers to training the neural network for one hour on a GPU. This is a common measurement, since NAS can be parallelized on multiple GPUs. Note that this measurement does not give an insight into what GPU was used. In recent years, there has been various methods applied to shorten the performance estimation to speed up runs. When using a performance estimator it is essential that it is correlated with the ground truth performance of an architecture and is robust towards different hyperparameter configurations and random initializations of the architecture. The estimation should not change much when different configurations or random seeds are used. Performance estimators can be grouped into different approaches as described in the following.

Model based methods consist of fully training the neural networks, obtaining $\theta^*(A)$ for many architectures $A \in \mathcal{A}$ to build a training dataset with samples $\{A, \theta^*(A)\}$. Then a model f is trained to predict $\theta^*(A)$ given A. While it takes a while to train neural networks and obtain a model f, query time is very fast. If f already exists, the evaluation of the search method becomes very fast. Representations of the architectures include graph neural networks [53, 75] and tree-based methods using the adjacency matrix as a feature [51, 85].

Learning curve based methods use partially trained models and often extrapolate the learning curve. This is done by fitting the partial learning curve to an ensemble of parametric models [71] or summing up losses over batches or epochs [18], sometimes weighting the epochs a certain way [72]. Early stopping, as used in our first work on NAS [37], is also a learning curve based method. We stop training early (after 30 epochs). However, on the largest dataset PEMS-BAY, runtime on four GPUs still takes over a month, resulting in thousands of GPU hours in total. Rorabaugh et al. [70] use an engine that can be plugged into existing NAS frameworks to decrease estimation time by up to 82 %. Their engine consists of two steps. First, the architecture is trained for a small amount and the final performance predicted. Then, a prediction analyser determines if a stable performance estimate was found.

Hybrid methods combine model based and learning curve based methods. These methods train models to predict $f(A)$ given A and the partial learning curve as features [6, 36].

One-shot estimators (OSE) refers to parameter sharing between all candidate solutions, since it requires the training cost of only one super-net. This method was first introduced in Efficient NAS by Pham et al. [65]. Each candidate solution is a subset of the super-net, sampled by taking different routes throughout the network. Since then, OSE is widely used in NAS [44, 47, 64, 83]. However, it has been shown that OSE can degrade the true ranking of the candidate solutions, reducing the effectiveness [73]. Since the aforementioned study was conducted on a small scale, Ning et al. [63] extend on this, finding that undertrained super-nets loose the ability to distinguish between close intra-level architectures. They propose several techniques to improve upon this, e.g. sampling variance and fairness improvements and dynamic search space pruning.

Zero-cost (ZC) proxies rank neural architectures within the search space without necessitating expensive training [1, 38, 40, 49, 50, 57, 80]. This is usually done by computing statistics based on the activations or outputs of one forward pass of a mini batch or the gradients of one backwards pass. They eliminate the need for costly training and have shown promising results across various domains, including image classification and natural language processing. However, they have not been thoroughly studied on regression tasks. Challenges related to weight initialization, mini-batch sampling and unclear correlation between ZC proxies and validation loss are also prevalent. In our second work on NAS [38] we explore the use of ZC proxies thoroughly. This includes large experiments regarding the robustness to hyperparameter configurations and random seeds for the most commonly used ZC proxies. We also conduct a study on the correlation of each ZC proxy with the validation loss after full training and use the best performing proxy in our evolutionary NAS method [37].

As presented, performance estimation in NAS is a complex, multi-faceted challenge. While traditional full training offers reliability, its computational expense has driven the advent of numerous alternative methods. Each of these approaches presents its own set of

trade-offs between accuracy and computational cost. Since ZC proxies have the potential to save the most computational resources, we extend the research on them for regression tasks, specifically traffic forecasting, in our work as one of the main contributions of this thesis.

4 Contributions and papers

In this chapter, I present a series of five papers written during my Ph.D., which collectively contribute to the development of a capable neural architecture search algorithm for real-world traffic forecasting. These papers build upon one another, showcasing the progress and the evolution of my research. I outline each paper below, highlighting their main ideas and contributions.

4.1 Enhanced K-Nearest Neighbor Model for Multi-step Traffic Flow Forecast in Urban Roads

A. Mallek, ***D. Klosa*** *and C. Büskens: Enhanced K-Nearest Neighbor Model For Multi-steps Traffic Flow Forecast in Urban Roads, 2022, (8th IEEE International Smart Cities Conference, IEEE Xplore, pp. 1-5)*

In this study, we introduce an enhanced k-nearest neighbor (EKNN) approach for accurately predicting short-term traffic flow in urban roads. Our objective is to develop an algorithm capable of effectively forecasting traffic flow volume, which is vital for intelligent transportation planning.

To accomplish this, we apply our EKNN model to 11 weeks of non-processed data collected from 7 inductive loop detectors installed on urban roads in downtown Bremen, made available by the Traffic Management Center (Verkehrmanagementzentrale, VMZ) Bremen. The training dataset consists of 8 weeks of data, while the remaining 3 weeks are used for testing. We evaluate the performance of our model across different day-hour categories, including rush hours.

The basic KNN model serves as a non-parametric data-driven approach that searches for historical patterns similar to the current state and generates predictions based on those patterns. To enhance the KNN model, we incorporate additional traffic attributes, such as detector-wise data and weekday-wise data. We investigate the impact of the state vector length, which determines how far back in time the data remains relevant for accurate predictions and introduce a search radius to select similar profiles for prediction.

To evaluate the accuracy of our EKNN model, we utilize the mean absolute error (MAE) and mean absolute percentage error (MAPE) metrics. The results demonstrate that the EKNN model achieves an average absolute relative error of 17% for 6-step (1-hour) predictions on the test set, excluding early hours with insignificant traffic. We find that a state vector length of 60 minutes (comprising 6 timestamps) yields the best results for our dataset.

In conclusion, our study contributes an EKNN algorithm for multi-step traffic flow prediction in urban roads. We emphasize the importance of considering additional traffic attributes, optimizing model parameters such as the state vector length and employing appropriate data processing techniques to improve prediction accuracy. However, we acknowledge the limitations of our study, including the absence of data on traffic lights. We recognize the need for further experimentation with larger and processed datasets.

4.2 Impact of Data Loss on Multi-Step Forecast of Traffic Flow in Urban Roads Using K-Nearest Neighbors

A. Mallek, ***D. Klosa*** *and C. Büskens: Impact of Data Loss on Multi-Step Forecast of Traffic Flow in Urban Roads Using K-Nearest Neighbors, 2022 (Sustainability 2022, 14, 11232)*

In this research, we explore the impact of data loss on the performance of the enhanced k-nearest (EKNN) model for multi-step traffic flow forecasting in urban roads. Our study utilizes real-world data obtained from seven inductive loop detectors provided by the Traffic Management Center (Verkehrsmanagementzentrale, VMZ) of Bremen.

To begin, we evaluate the performance of the EKNN model on a complete dataset spanning 11 weeks. This serves as a benchmark for assessing the model's accuracy in the absence of data loss. We then proceed to artificially generate 50 incomplete datasets, each with varying gap sizes from a few timestamps to multiple weeks and completeness levels (50% to 90%). This allows us to simulate different scenarios of data loss and analyse the behaviour of the EKNN model under these conditions.

To address the challenge of data loss, we propose three computationally-efficient techniques for reconstructing the missing parts of the incomplete datasets. They include averaging methods and linear regression.

Next, we assess the performance of EKNN on the original dataset, incomplete datasets and reconstructed datasets. Our experimental results demonstrate the effectiveness of

the EKNN model in accurately forecasting traffic flow. The accuracy of the model varies depending on the gap lengths and completeness levels of the datasets. We observe that the EKNN model achieves an average accuracy of 83% in six-step forecasts over a three-week test set when applied to the original dataset. Additionally, we find that the performance of the model improves when applied to the reconstructed datasets compared to the incomplete ones.

Overall, our research provides valuable insights into the behaviour of the EKNN model in the presence of data loss and imputation of missing data. These findings contribute to the development of robust data-driven models in the field of intelligent transportation.

4.3 Short-Term Traffic Flow Forecast Using Regression Analysis and Graph Convolutional Neural Networks

***D. Klosa**, A. Mallek and C. Büskens: Short-Term Traffic Flow Forecast Using Regression Analysis and Graph Convolutional Neural Networks, 2021, (7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys), Haikou, Hainan, China, 2021, pp. 1413-1418)*

In this paper, we present our research on short-term traffic flow forecasting in urban arterial roads in Bremen. We propose linear regression and graph convolutional neural networks (GCNN) to forecast traffic flow. The models are applied to 11 weeks of real-data collected from 7 loop detectors installed in downtown Bremen, with 3 weeks dedicated to testing their performance.

The linear regression model captures linear relationships between variables using polynomial regression. We utilize different time frames to learn the linear dependencies between data points. The model is trained separately for each week-day, with a polynomial of degree 10 for the daily regression and a polynomial of degree 5 for the hourly regression. The predicted flow values are calculated based on the regression polynomials. A correction step is applied to improve accuracy by combining regression values with data from the previous week.

The GCNN model is based on the global spatial-temporal graph convolutional network (GSTGCN), employing dilated 1D convolutions to learn short-term and long-term relationships in the temporal space. The model incorporates a graph convolution that considers the distance between sensors and a global correlated spatial mechanism based on the connectivity within the traffic network. The features from different time frames

are fused and passed through a fully connected neural network for prediction. The GCNN model predicts traffic flow by considering data from all detectors simultaneously.

We evaluate the prediction accuracy of both models using mean absolute percentage error (MAPE) and mean absolute error (MAE) metrics. The experimental results show that both models are competitive, with an average MAPE of around 19% during morning and evening peak times and an average MAPE of around 25% for all-day hours. The accuracy varies across detectors, mainly depending on the flow volume at each location.

In conclusion, our study focuses on short-term traffic flow forecasting in a congested area of Bremen. The proposed linear regression and GCNN models provide satisfactory accuracy in predicting traffic flow. Future directions include hybridizing the models to leverage their strengths and incorporating traffic light data to improve predictions of fluctuating traffic patterns.

4.4 Evolutionary Neural Architecture Search for Traffic Forecasting

***D. Klosa** and C. Büskens: Evolutionary Neural Architecture Search for Traffic Forecasting, 2022, (In Proceedings of the 21st IEEE International Conference on Machine Learning and Applications, pp. 1230-1237)*

In our research, we investigate the application of evolutionary neural architecture search (ENAS) using a genetic algorithm (GA) for traffic forecasting. While current research primarily relies on manually constructing neural network architectures without the aid of neural architecture search (NAS), we aim to explore the potential of ENAS for finding optimal neural network architectures to achieve accurate traffic prediction.

Our approach involves deploying a GA to search for the best neural network architectures capable of predicting traffic conditions. The search space for the GA consists of various combinations of none, skip connections, dilated convolutions and graph convolutions. We initialize a population of neural architectures and evaluate their performance on traffic prediction benchmarks. The architectures are trained for a limited number of epochs to estimate their performance. The best-performing architectures are selected for further evolution. Through crossover and mutation operations, we create offspring architectures with potential improvements. This process continues for multiple generations. The algorithm terminates after a predefined number of cycles. Then, we select the best-performing architecture as our final model.

To evaluate the effectiveness of our ENAS framework, we conduct experiments on four

real-world traffic datasets recorded in California. We compare the performance of our framework against several baseline models, including historical average (HA), AGCRN, Graph WaveNet and AutoSTG.

Our experimental results demonstrate that the architectures obtained through the GA-based ENAS approach can match or even surpass the state-of-the-art models on the traffic prediction benchmarks. Specifically, our framework outperforms other methods on the PeMSD4 and PeMSD8 datasets while achieving competitive results on the others. We also conduct an ablation study to analyse the impact of dilated convolution and graph convolution in the architectures, with the results highlighting the importance of both operations in improving performance.

In conclusion, our ENAS framework presents a promising approach for traffic forecasting by automating the search for optimal neural network architectures. The experimental results validate the effectiveness of our approach on real-world traffic prediction benchmarks. We also discuss potential future directions, including exploring alternative optimization algorithms, adaptive adjacency matrices and the incorporation of additional historical data for prediction.

4.5 Low cost evolutionary neural architecture search applied to traffic forecasting

D. Klosa *and C. Büskens: Low Cost Evolutionary Neural Architecture Search (LENAS) Applied to Traffic Forecasting, 2023, (Mach. Learn. Knowl. Extr. 2023, 5, 830-846)*

In this paper, we extend the existing research on evolutionary NAS (ENAS) [37] and explore the use of zero-cost (ZC) proxies to estimate the performance of neural network architectures without the need for time-consuming training. ZC proxies offer a cost-effective solution to the performance estimation problem in NAS, potentially accelerating the search process. The investigated ZC proxies require just one forward propagation of inputs and sometimes one backwards propagation for evaluation and therefore speed-up the performance estimation tremendously.

Our research introduces ZC proxies as a means to evaluate network architectures in the ENAS framework. We evaluate the stability of ZC proxies with respect to weight initialization, mini-batch size and sampling and architecture size. This is done by scoring the same architectures with different hyperparameter combinations and random seeds and comparing spearman rank correlation between those runs. ZC proxies leading to high correlation are robust since the ranking of architectures is similar across runs. Among the ZC proxies examined, we identify naswot as the most robust with a perfect

spearman rank correlation, making it a suitable choice for the traffic forecasting task.

To evaluate the performance estimation properties of the ZC proxies, we conduct experiments on the same traffic speed and traffic flow benchmarks as in [37]. For this, we compute spearman rank correlation between validation loss after full training with the ZC proxy score. Again, naswot performs the best with a correlation of 0,737. Although not perfect, the results demonstrate that utilizing naswot scores instead of training to convergence can significantly accelerate the neural architecture search process for a slight decrease in performance of the final architecture.

In conclusion, our research contributes to the field of NAS for traffic forecasting by exploring the application of ZC proxies to speed up search times. We can greatly increase the speed of our ENAS framework making it viable to use in traffic control centers without the need for expensive hardware. We discuss further research directions, e.g. towards better ZC proxies designed for regression tasks like traffic prediction.

5 Summary and future work

5.1 Summary

In the first part of this thesis, we have established the foundational concepts for the papers we have published. We gave an introduction into the task of traffic prediction in Chapter 1. We have shown a real world example, described the available data, the challenges that come with it and the potential benefits of accurate traffic predictions. In Chapter 2, we went over the fundamentals of machine learning in the form of supervised learning for regression. We briefly introduced two classical machine learning models, polynomial regression and k-nearest neighbours and detailed the inner workings of (convolutional) neural networks and how they are trained. In Chapter 3, we introduced neural architecture search for the task of traffic prediction and described our general framework. This chapter, in conjunction with the accompanying published papers, highlights the authors primary contribution: the development of an evolutionary neural architecture search approach that identifies a viable solution for the task of traffic prediction.

5.2 Future work

Neural architecture search is a fast growing field within deep learning. However, as most of the research is focused on computer vision and natural language processing, regression is falling short in current research. The intersection of NAS and traffic prediction is very limited. Further intersecting with evolutionary algorithms as search methods yields two published papers (by us). We use a simple genetic algorithm without any efforts towards hyperparameter tuning of this algorithm. This should be done in future research and additionally compared to other evolutionary algorithms. Intersecting with zero-cost proxies results in one published paper (by us). All of the currently used zero-cost proxies are engineered with classification in mind. The best performing proxy, naswot [57], operates under the assumption that samples in a mini-batch should activate different regions of the neural network. However, this assumption proves to be invalid for regression tasks. As demonstrated in Chapter 1, timeseries data, particularly for flow, can exhibit significant similarity, undermining the effectiveness of this approach.

This results in subpar correlation between the proxy and the validation loss. Future research should focus on engineering zero-cost proxies specifically for regression tasks. Again, looking at NAS for classification, there are benchmarks comprising the dataset and the search space such that search methods can easily be compared. This lacks in traffic prediction and efforts should be made towards creating such benchmarks.

Bibliography

[1] Abdelfattah, Mohamed S., Mehrotra, Abhinav, Dudziak, Lukasz, and Lane, Nicholas D. "Zero-Cost Proxies for Lightweight NAS". *CoRR* abs/2101.08134 (2021). arXiv: 2101.08134.

[2] Ashok, Anubhav, Rhinehart, Nicholas, Beainy, Fares, and Kitani, Kris M. "N2N Learning: Network to Network Compression via Policy Gradient Reinforcement Learning". *CoRR* abs/1709.06030 (2017). arXiv: 1709.06030.

[3] Awad, Noor H., Mallik, Neeratyoy, and Hutter, Frank. "Differential Evolution for Neural Architecture Search". *CoRR* abs/2012.06400 (2020). arXiv: 2012.06400.

[4] Bai, Lei, Yao, Lina, Li, Can, Wang, Xianzhi, and Wang, Can. "Adaptive Graph Convolutional Recurrent Network for Traffic Forecasting". *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS'20. Vancouver, BC, Canada: Curran Associates Inc., 2020.

[5] Baker, Bowen, Gupta, Otkrist, Naik, Nikhil, and Raskar, Ramesh. "Designing Neural Network Architectures using Reinforcement Learning". *CoRR* abs/1611.02167 (2016). arXiv: 1611.02167.

[6] Baker, Bowen, Gupta, Otkrist, Raskar, Ramesh, and Naik, Nikhil. "Accelerating Neural Architecture Search using Performance Prediction" (2017). arXiv: 1705.10823 [cs.LG].

[7] Baker, Bowen, Gupta, Otkrist, Raskar, Ramesh, and Naik, Nikhil. "Practical Neural Network Performance Prediction for Early Stopping". *CoRR* abs/1705.10823 (2017). arXiv: 1705.10823.

[8] Bellman, R. *Adaptive Control Processes: A Guided Tour*. Princeton Legacy Library. Princeton University Press, 1961.

[9] Bellman, R., Corporation, Rand, and Collection, Karreman Mathematics Research. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1957.

[10] Bello, Irwan, Zoph, Barret, Vasudevan, Vijay, and Le, Quoc V. "Neural Optimizer Search with Reinforcement Learning". *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, Aug. 2017, pp. 459–468.

[11] Bergstra, James and Bengio, Yoshua. "Random Search for Hyper-Parameter Optimization." *Journal of Machine Learning Research* 13 (2012), pp. 281–305.

[12] Bruna, Joan, Zaremba, Wojciech, Szlam, Arthur, and Lecun, Yann. "Spectral Networks and Locally Connected Networks on Graphs" (Dec. 2013).

[13] Chen, Donghui, Chen, Ling, Shang, Zongjiang, Zhang, Youdong, Wen, Bo, and Yang, Chenghu. "Scale-Aware Neural Architecture Search for Multivariate Time Series Forecasting". *CoRR* abs/2112.07459 (2021). arXiv: `2112.07459`.

[14] Chen, Xin, Xie, Lingxi, Wu, Jun, and Tian, Qi. "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation". *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 1294–1303.

[15] Clerc, Maurice. "Discrete Particle Swarm Optimization, illustrated by the Traveling Salesman Problem". *New Optimization Techniques in Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 219–239.

[16] Defferrard, Michaël, Bresson, Xavier, and Vandergheynst, Pierre. "Convolutional neural networks on graphs with fast localized spectral filtering". *Advances in Neural Information Processing Systems* (2016). Cited by: 4898, pp. 3844–3852.

[17] Diao, Zulong, Wang, Xin, Zhang, Dafang, Liu, Yingru, Xie, Kun, and He, Shaoyao. "Dynamic Spatial-Temporal Graph Convolutional Neural Networks for Traffic Forecasting". *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (July 2019), pp. 890–897.

[18] Domhan, Tobias, Springenberg, Jost Tobias, and Hutter, Frank. "Speeding up Automatic Hyperparameter Optimization of Deep Neural Networks by Extrapolation of Learning Curves". *Proceedings of the 24th International Conference on Artificial Intelligence*. IJCAI'15. Buenos Aires, Argentina: AAAI Press, 2015, pp. 3460–3468.

[19] Eiben, A. and Schippers, C. "On Evolutionary Exploration and Exploitation". *Fundam. Inform.* 35 (Aug. 1998), pp. 35–50.

[20] Ge, Liang, Li, Siyu, Wang, Yaqian, Chang, Feng, and Wu, Kunyan. "Global Spatial-Temporal Graph Convolutional Network for Urban Traffic Speed Prediction". *Applied Sciences* 10.4 (2020).

[21] Goldberg, D.E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley series in artificial intelligence. Addison-Wesley, 1989.

[22] Gomez, Faustino J. and Miikkulainen, Risto. "Solving Non-Markovian Control Tasks With Neuroevolution". *Proceedings of the International Joint Conference on Artificial Intelligence*. San Francisco, CA: Kaufmann, 1999, pp. 1356–1361.

[23] Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[24] Gruau, Frédéric, Whitley, Darrell, and Pyeatt, Larry. "A comparison between cellular encoding and direct encoding for genetic neural networks". *Proceedings of the 1st Annual Conference on Genetic Programming*. Stanford, California: MIT Press, 1996, pp. 81–89.

[25] Hammond, David K., Vandergheynst, Pierre, and Gribonval, Rémi. "Wavelets on graphs via spectral graph theory". *Applied and Computational Harmonic Analysis* 30.2 (2011), pp. 129–150.

[26] Hassanat, Ahmad, Almohammadi, Khalid, Alkafaween, Esra'a, Abunawas, Eman, Hammouri, Awni, and Prasath, V. B. Surya. "Choosing Mutation and Crossover Ratios for Genetic Algorithms—A Review with a New Dynamic Approach". *Information* 10.12 (2019).

[27] He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. "Deep Residual Learning for Image Recognition". *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385.

[28] He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. "Identity Mappings in Deep Residual Networks". *CoRR* abs/1603.05027 (2016). arXiv: 1603.05027.

[29] Henaff, Mikael, Bruna, Joan, and LeCun, Yann. "Deep Convolutional Networks on Graph-Structured Data". *CoRR* abs/1506.05163 (2015). arXiv: 1506.05163.

[30] Holland, John H. "Genetic Algorithms". *Scientific American* 267.1 (1992), pp. 66–73.

[31] Inrix. *INRIX 2022 Traffic Scorecard Report*.

[32] Ioffe, Sergey and Szegedy, Christian. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167.

[33] Ke, Songyu, Pan, Zheyi, He, Tianfu, Liang, Yuxuan, Zhang, Junbo, and Zheng, Yu. "AutoSTG+: An automatic framework to discover the optimal network for spatio-temporal graph prediction". *Artificial Intelligence* 318 (2023), p. 103899.

[34] Kennedy, J. and Eberhart, R. "Particle swarm optimization". *Proceedings of ICNN'95 - International Conference on Neural Networks*. Vol. 4. 1995, 1942–1948 vol.4.

[35] Kipf, Thomas N. and Welling, Max. "Semi-Supervised Classification with Graph Convolutional Networks". *CoRR* abs/1609.02907 (2016). arXiv: 1609.02907.

[36] Klein, Aaron, Falkner, Stefan, Springenberg, Jost Tobias, and Hutter, Frank. "Learning Curve Prediction with Bayesian Neural Networks". *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[37] Klosa, Daniel and Büskens, Christof. "Evolutionary Neural Architecture Search for Traffic Forecasting". *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*. 2022, pp. 1230–1237.

[38] Klosa, Daniel and Büskens, Christof. "Low Cost Evolutionary Neural Architecture Search (LENAS) Applied to Traffic Forecasting". *Machine Learning and Knowledge Extraction* 5.3 (2023), pp. 830–846.

[39] Klosa, Daniel, Mallek, Amin, and Büskens, Christof. "Short-Term Traffic Flow Forecast Using Regression Analysis and Graph Convolutional Neural Networks". *2021 IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*. 2021, pp. 1413–1418.

[40] Krishnakumar, Arjun, White, Colin, Zela, Arber, Tu, Renbo, Safari, Mahmoud, and Hutter, Frank. "NAS-Bench-Suite-Zero: Accelerating Research on Zero Cost Proxies" (2022). arXiv: 2210.03230 [cs.LG].

[41] LeCun, Yann, Bengio, Y., and Hinton, Geoffrey. "Deep Learning". *Nature* 521 (May 2015), pp. 436–44.

[42] Li, Qimai, Han, Zhichao, and Wu, Xiao-Ming. "Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning". *CoRR* abs/1801.07606 (2018). arXiv: 1801.07606.

[43] Li, Yaguang, Yu, Rose, Shahabi, Cyrus, and Liu, Yan. "Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting". *International Conference on Learning Representations*. 2018.

[44] Liang, Hanwen, Zhang, Shifeng, Sun, Jiacheng, He, Xingqiu, Huang, Weiran, Zhuang, Kechen, and Li, Zhenguo. "DARTS+: Improved Differentiable Architecture Search with Early Stopping". *CoRR* abs/1909.06035 (2019). arXiv: 1909.06035.

[45] Liu, Chenxi, Zoph, Barret, Shlens, Jonathon, Hua, Wei, Li, Li-Jia, Fei-Fei, Li, Yuille, Alan L., Huang, Jonathan, and Murphy, Kevin. "Progressive Neural Architecture Search". *CoRR* abs/1712.00559 (2017). arXiv: 1712.00559.

[46] Liu, Hanxiao, Simonyan, Karen, Vinyals, Oriol, Fernando, Chrisantha, and Kavukcuoglu, Koray. "Hierarchical Representations for Efficient Architecture Search". *CoRR* abs/1711.00436 (2017). arXiv: 1711.00436.

[47] Liu, Hanxiao, Simonyan, Karen, and Yang, Yiming. "DARTS: Differentiable Architecture Search". *arXiv preprint arXiv:1806.09055* (2018).

[48] Liu, Xiaobo, Zhang, Chaochao, Cai, Zhihua, Yang, Jianfeng, Zhou, Zhilang, and Gong, Xin. "Continuous Particle Swarm Optimization-Based Deep Learning Architecture Search for Hyperspectral Image Classification". *Remote Sensing* 13.6 (2021).

[49] Lopes, Vasco, Alirezazadeh, Saeid, and Alexandre, Luís A. "EPE-NAS: Efficient Performance Estimation Without Training for Neural Architecture Search". *Artificial Neural Networks and Machine Learning – ICANN 2021* (2021), pp. 552–563.

[50] Lopes, Vasco, Santos, Miguel, Degardin, Bruno, and Alexandre, Luís A. "Guided Evolution for Neural Architecture Search". *CoRR* abs/2110.15232 (2021). arXiv: 2110.15232.

[51] Luo, Renqian, Tan, Xu, Wang, Rui, Qin, Tao, Chen, Enhong, and Liu, Tie-Yan. "Neural Architecture Search with GBDT". *CoRR* abs/2007.04785 (2020). arXiv: 2007.04785.

[52] Luo, Renqian, Tan, Xu, Wang, Rui, Qin, Tao, Chen, Enhong, and Liu, Tie-Yan. "Semi-Supervised Neural Architecture Search". *CoRR* abs/2002.10389 (2020). arXiv: 2002.10389.

[53] Ma, Lizheng, Cui, Jiaxu, and Yang, Bo. "Deep Neural Architecture Search with Deep Graph Bayesian Optimization". *2019 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*. 2019, pp. 500–507.

[54] Mallat, Stéphane. *A wavelet tour of signal processing*. Elsevier, 1999.

[55] Mallek, Amin, Klosa, Daniel, and Büskens, Christof. "Enhanced K-Nearest Neighbor Model For Multi-steps Traffic Flow Forecast in Urban Roads". *2022 IEEE International Smart Cities Conference (ISC2)*. 2022, pp. 1–5.

[56] Mallek, Amin, Klosa, Daniel, and Büskens, Christof. "Impact of Data Loss on Multi-Step Forecast of Traffic Flow in Urban Roads Using K-Nearest Neighbors". *Sustainability* 14.18 (2022).

[57] Mellor, Joseph, Turner, Jack, Storkey, Amos, and Crowley, Elliot J. "Neural Architecture Search without Training" (2020).

[58] Miller, Brad L. and Goldberg, David E. "Genetic Algorithms, Tournament Selection, and the Effects of Noise". *Complex Syst.* 9 (1995).

[59] Mitchell, T.M. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997.

[60] Mun, Jiwoo, Ha, Seokhyeon, and Lee, Jungwoo. "DE-DARTS: Neural architecture search with dynamic exploration". *ICT Express* 9.3 (2023), pp. 379–384.

[61] Na, Hu, Dafang, Zhang, Kun, Xie, Wei, Liang, and Meng-Yen, Hsieh. "Graph learning-based spatial-temporal graph convolutional neural networks for traffic forecasting". *Connection Science* 34.1 (2022), pp. 429–448. eprint: https://doi.org/10.1080/09540091.2021.2006607.

[62] Nilsson, N. *Introduction to Machine Learning*. MIT Press (to apprear), 1998.

[63] Ning, Xuefei, Li, Wenshuo, Zhou, Zixuan, Zhao, Tianchen, Zheng, Yin, Liang, Shuang, Yang, Huazhong, and Wang, Yu. "A Surgery of the Neural Architecture Evaluators". *CoRR* abs/2008.03064 (2020). arXiv: 2008.03064.

[64] Pan, Zheyi, Ke, Songyu, Yang, Xiaodu, Liang, Yuxuan, Yu, Yong, Zhang, Junbo, and Zheng, Yu. "AutoSTG: Neural Architecture Search for Predictions of Spatio-Temporal Graph". Apr. 2021, pp. 1846–1855.

[65] Pham, Hieu, Guan, Melody, Zoph, Barret, Le, Quoc, and Dean, Jeff. "Efficient Neural Architecture Search via Parameters Sharing". *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, July 2018, pp. 4095–4104.

[66] Rahimipour, Shiva, Moienfar, Ray, and Hashemi, S. Mehdi. "Traffic Prediction Using a Self-Adjusted Evolutionary Neural Network". *Journal of Modern Transportation* 27 (Dec. 2018).

[67] Ratner, Alexander J, Ehrenberg, Henry, Hussain, Zeshan, Dunnmon, Jared, and Ré, Christopher. "Learning to Compose Domain-Specific Transformations for Data Augmentation". *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017.

[68] Real, Esteban, Aggarwal, Alok, Huang, Yanping, and Le, Quoc V. "Regularized Evolution for Image Classifier Architecture Search". *CoRR* abs/1802.01548 (2018). arXiv: 1802.01548.

[69] Real, Esteban, Moore, Sherry, Selle, Andrew, Saxena, Saurabh, Suematsu, Yutaka Leon, Tan, Jie, Le, Quoc V., and Kurakin, Alexey. "Large-scale evolution of image classifiers". *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 2902–2911.

[70] Rorabaugh, Ariel Keller, Caíno-Lores, Silvina, II, Michael R. Wyatt, Johnston, Travis, and Taufer, Michela. "PEng4NN: An Accurate Performance Estimation Engine for Efficient Automated Neural Network Architecture Search". *CoRR* abs/2101.04185 (2021). arXiv: 2101.04185.

[71] Ru, Binxin, Lyle, Clare, Schut, Lisa, Wilk, Mark, and Gal, Yarin. "Revisiting the Train Loss: an Efficient Performance Estimator for Neural Architecture Search" (June 2020).

[72] Ru, Robin, Lyle, Clare, Schut, Lisa, Fil, Miroslav, Wilk, Mark van der, and Gal, Yarin. "Speedy Performance Estimation for Neural Architecture Search". *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan. Vol. 34. Curran Associates, Inc., 2021, pp. 4079–4092.

[73] Sciuto, Christian, Yu, Kaicheng, Jaggi, Martin, Musat, Claudiu, and Salzmann, Mathieu. "Evaluating the Search Phase of Neural Architecture Search". *CoRR* abs/1902.08142 (2019). arXiv: 1902.08142.

[74] Shi, Han, Pi, Renjie, Xu, Hang, Li, Zhenguo, Kwok, James T., and Zhang, Tong. "Multi-objective Neural Architecture Search via Predictive Network Performance Optimization". *CoRR* abs/1911.09336 (2019). arXiv: 1911.09336.

[75] Shi, Han, Pi, Renjie, Xu, Hang, Li, Zhenguo, Kwok, James T., and Zhang, Tong. "Bridging the Gap between Sample-based and One-shot Neural Architecture Search with BONAS". *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. 2020.

[76] Shuman, David I., Narang, Sunil K., Frossard, Pascal, Ortega, Antonio, and Vandergheynst, Pierre. "Signal Processing on Graphs: Extending High-Dimensional Data Analysis to Networks and Other Irregular Data Domains". *CoRR* abs/1211.0053 (2012). arXiv: 1211.0053.

[77] Stanley, Kenneth O. and Miikkulainen, Risto. "Evolving Neural Networks Through Augmenting Topologies". *Evolutionary Computation* 10.2 (2002), pp. 99–127.

[78] Storn, Rainer and Price, Kenneth. "Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces". *Journal of Global Optimization* 11 (Jan. 1997), pp. 341–359.

[79] Tang, Cong, Sun, Jingru, Sun, Yichuang, Peng, Mu, and Gan, Nianfei. "A General Traffic Flow Prediction Approach Based on Spatial-Temporal Graph Attention". *IEEE Access* 8 (2020), pp. 153731–153741.

[80] White, Colin, Zela, Arber, Ru, Binxin, Liu, Yang, and Hutter, Frank. "How Powerful are Performance Predictors in Neural Architecture Search?" (2021).

[81] Wu, Zonghan, Pan, Shirui, Chen, Fengwen, Long, Guodong, Zhang, Chengqi, and Yu, Philip S. "A Comprehensive Survey on Graph Neural Networks". *CoRR* abs/1901.00596 (2019). arXiv: 1901.00596.

[82] Yan, X. *Linear Regression Analysis: Theory and Computing*. World Scientific Publishing Company Pte Limited, 2009.

[83] Yang, Zhaohui, Wang, Yunhe, Chen, Xinghao, Shi, Boxin, Xu, Chao, Xu, Chunjing, Tian, Qi, and Xu, Chang. "CARS: Continuous Evolution for Efficient Neural Architecture Search". *CoRR* abs/1909.04977 (2019). arXiv: 1909.04977.

[84] Yu, Bing, Yin, Haoteng, and Zhu, Zhanxing. "Spatio-temporal Graph Convolutional Neural Network: A Deep Learning Framework for Traffic Forecasting". *CoRR* abs/1709.04875 (2017). arXiv: 1709.04875.

[85] Zela, Arber, Siems, Julien N., Zimmer, Lucas, Lukasik, Jovita, Keuper, Margret, and Hutter, Frank. "Surrogate NAS Benchmarks: Going Beyond the Limited Search Spaces of Tabular NAS Benchmarks". *International Conference on Learning Representations*. 2020.

[86] Zhang, Junping, Wang, Fei-Yue, Wang, Kunfeng, Lin, Wei-Hua, Xu, Xin, and Chen, Cheng. "Data-Driven Intelligent Transportation Systems: A Survey". *IEEE Transactions on Intelligent Transportation Systems* 12.4 (2011), pp. 1624–1639.

[87] Zhang, Si, Tong, Hanghang, Xu, Jiejun, and Maciejewski, Ross. "Graph convolutional networks: a comprehensive review". *Computational Social Networks* 6 (Nov. 2019).

[88] Zhou, Jie, Cui, Ganqu, Zhang, Zhengyan, Yang, Cheng, Liu, Zhiyuan, and Sun, Maosong. "Graph Neural Networks: A Review of Methods and Applications". *CoRR* abs/1812.08434 (2018). arXiv: 1812.08434.

[89] Zhou, Kuangqi, Dong, Yanfei, Lee, Wee Sun, Hooi, Bryan, Xu, Huan, and Feng, Jiashi. "Effective Training Strategies for Deep Graph Neural Networks". *ArXiv* abs/2006.07107 (2020).

[90] Zoph, Barret and Le, Quoc V. "Neural Architecture Search with Reinforcement Learning" (2016). arXiv: 1611.01578 [cs.LG].

[91] Zoph, Barret, Vasudevan, Vijay, Shlens, Jonathon, and Le, Quoc V. "Learning Transferable Architectures for Scalable Image Recognition". *CoRR* abs/ 1707.07012 (2017). arXiv: 1707.07012.

Enhanced K-Nearest Neighbor Model For Multi-steps Traffic Flow Forecast in Urban Roads

Amin Mallek
Center for Industrial Mathematics
University of Bremen
Bremen, Germany
amallek@uni-bremen.de

Daniel Klosa
Center for Industrial Mathematics
University of Bremen
Bremen, Germany
dklosa@uni-bremen.de

Christof Büskens
Center for Industrial Mathematics
University of Bremen
Bremen, Germany
bueskens@math.uni-bremen.de

***Abstract*—Short-term flow forecast is a fundamental key in intelligent transportation planning. Often accurate predictions are provided by the predictive models the most adapted to the nature of the addressed problem. In this paper we present a k-Nearest Neighbor approach (E-KNN) enhanced by taking advantage of traffic attributes. The proposed model is applied to 11 weeks of non-processed data, recorded by 7 inductive loop detectors installed on urban roads located in downtown of Bremen (Germany). The performance of E-KNN is tested on 3 weeks of data and reported following different day-hours categories, including rush hours. Excluding early day-hours where traffic is insignificant, E-KNN performs 6-steps (1h) prediction with an average absolute relative error of 17% on test-set.**

***Index Terms*—Traffic flow; K-Nearest Neighbor; Short-term forecast; Urban roads; Multi-step prediction; Machine learning.**

I. Introduction

Huge part of business and economy nowadays heavily rely on transportation systems. Among others, e-commerce, which is partly based on delivering goods to customers, and transportation of individuals from and to work for example. Optimizing costs and time for such operations require efficient and intelligent transportation management systems. One of the crucial components of these systems is the prediction of different attributes related to traffic, especially traffic flow volume. Often, this latter is the main element on which other traffic-related features are based, and is a vital topic discussed in both academia and industry. Traffic managers usually depend on short-term traffic flow forecasts to plan and formulate efficient strategies in order to alleviate road congestion and further optimize vehicular traffic inside cities. Moreover, travelers as well refer to these forecasts to take decisions about their traveling plans. The development of approaches for the purpose of accurate short-term flow forecasting might not be successful without a large amount of data. Therefore, traffic management centers deploy a large range of tools to monitor and record traffic attributes, including, inductive loop detectors, video and image processing, radars of different kinds and other Internet of Things (IoT) mechanisms. Recently and in the past years, researchers have extensively examined the problem of short-term traffic flow forecast. Consequently, several data-driven models have been proposed to tackle this problem, categorized into two main classes: parametric and non-parametric models. We can briefly describe parametric models as the models that output predictions based on an explicit function defined within a finite set of parameters. These parameters are often estimated by training the model with a given dataset, for instance: ARIMA and its variants [6], [9], [16], Neural Networks [1], [11], [13], deep learning [12], Linear Regression [8]. In contrast, non-parametric methods deliver predictions without assuming any prior knowledge or having any explicit formulas, such as Support Vector Regression (SVR) [3], [10] and k-Nearest Neighbor [2], [4], [17].

In this paper we address the problem of multi-steps short-term flow volume forecast in urban roads. This problem is part of DiSCO2 project conducted at the University of Bremen. The aim of this project is to set the stage for decision makers to take actions to reduce CO2 emissions in Bremen (Germany). To this end, we develop an enhanced k-Nearest Neighbor model based on traffic features. This method (shortened as KNN), is known to be a non-parametric data-driven model and has extensively been investigated in the literature. Old KNN models mainly focus on single step forecasts [5], [14], [15]. However, this technique and other non-parametric models have the advantage of being flexible and easily extensible. Therefore, herein we extend the classical k-Nearest Neighbor to what we call enhanced KNN referred to as E-KNN model. This latter takes into account more attributes related to traffic to improve forecasting accuracy. Several improvements to KNN model have already been considered in the context of traffic flow in some papers, including [2] that deployed a weighted Gaussian method to compute forecasts instead of the typical ones as in [15]. The authors in [17] incorporated a time constraint in neighbor selection and a minima distance to avoid selection of highly auto-correlated candidates. Cheng

Funded by the European Regional Development Fund (ERDF).

European Union
Investing in Bremen's Future
European Regional
Development Fund

et al. [4] developed a KNN model based on the assumption that traffic between adjacent road segments within assigned time periods is not correlated. This saptiotemporal approach comprehensively considers the spatial heterogeneity of traffic. Our E-KNN takes into account a search radius to ensure that selected profiles share similar characteristics. It assumes as well that the flow is not only distinct between weekends and working days, bus also among all weekdays. Usually studies are carried out on processed or filtered data, herein we measure the performances of our designed technique using raw data, provided by the Traffic Management Center (VMZ) of Bremen, with no preprocessing, meaning that noise, corrupted data and outliers are kept as they are in our dataset. The purpose behind this is to have an idea about the accuracy of the model when it is operated online, as needed in our project. Furthermore, due to the same reason, the model performs 6-steps (1h) forecasts at once.

The rest of this of this paper is organized as follows. In section II we thoroughly describe the basic k-Nearest Neighbor model, then the E-KNN approach. Section II introduces in details our dataset. Afterwards, the accuracy of E-KNN is experimentally tested and reported in Section IV. Finally, the paper is concluded with future directions in Section V.

II. Methodology

k-Nearest Neighbor model is one of the famous data-driven predictive models applied to different problems present in the literature. In its essence, this approach explores historical data to fetch for patterns in the past similar to a present one. The algorithm then tries to generate forecasts based on the future states of the historical patterns that are hypothetically closely similar to a current state. The quality of this model as any other data-driven model depends on how big the data is and also how well it is represented. In what follows, we detail the basic components of KNN and how they are characterized. Afterwards, based on the nature of our problem, we integrate in the model some enhancements to improve its performance.

1) Basic KNN and notations: In this subsection we introduce the set of notations used to describe our KNN model and its functionalities. First of all, traffic flow volume at instant t at some detector (sensor) d is denoted by $f_d(t)$. Thus, a flow volume series at some detector d in a given time frame between t_i and t_j $(i \leq j)$ is defined by the following vector:

$$v_d(t_i, t_j) = [f_d(t_i), f_d(t_{i+1}), \ldots, f_d(t_{j-1}), f_d(t_j)] \quad (1)$$

For simplicity, we concisely write $f(t)$ and $v(t_i, t_j)$, unless the detector is required to be mentioned. We define a state vector of length l at instant t as a vector of flow volume comprising the traffic flow from instant t backwards to instant $t - l$, therefore it can be seen as:

$$v(t-l, t) = [f(t-l), f(t-l+1), \ldots, f(t-1), f(t)] \quad (2)$$

The forecast of the next s steps are given in a prediction vector denoted by v' and expressed as follows:

$$v'(t+1, t+s) = [f'(t+1), f'(t+2), \ldots, f'(t+s-1), f'(t+s)] \quad (3)$$

Once the state vector is defined, the other components of KNN framework can then be set. The first task is to select a set of nearest neighbors to a given state vector based on measuring a certain distance between this latter and other candidate vectors. The best k candidates (neighbors) are then selected to be considered in the prediction process. Various methods are usually used whether to determine the closeness of current state vector to other vectors or to produce forecasts. The distance between two state vectors is commonly given by the Euclidean distance, however sometimes and in some cases correlation coefficient distance is also considered where its superiority is proven [17]. Averaging the k nearest neighbors is often the common way used to predict future states. Though, many other approaches were also applied in the literature to obtain forecasts, including weighting the k nearest neighbors according to their distance to current state vector [7] and Gaussian-weight distance [2], [4].

Our KNN algorithm is designed to perform multi-steps prediction, therefore, at some instant t, the vector aggregating current state and s future steps can be seen as follows:

$$E(t) = v(t-l, t) + v'(t+1, t+s) = [f(t-l), \ldots, f(t), f'(t+1), \ldots, f'(t+s)] \quad (4)$$

In our study, we use Euclidean distance to rank the neighbors of state vectors. Firstly, because the correlation coefficient distance proved to be inferior to Euclidean distance through the experiments. Secondly, the Gaussian-based distance did not increase the accuracy of the predictions. We also use simple averaging of the k nearest neighbors. Note that weighting of neighbors procedure has also been tested and gave the same results as simple averaging. Mathematically, the Euclidean distance, in our case, is given as follows:

$$dist^{(i,j)}(v^i, v^j) = \sum_{\lambda=0}^{l} (f^i(t-\lambda) - f^j(t-\lambda))^2 \quad (5)$$

such that v^i and v^j are two state vectors. As said before, our KNN is designed to forecast multi-steps, hence the formula to deliver forecast vector is:

$$v'(t+1, t+s) = [f'(t+1), \ldots, f'(t+s)] = [\sum_{i=1}^{k} f^{(i)}(t+1)/k, \ldots, \sum_{i=1}^{k} f^{(i)}(t+s)/k] \quad (6)$$

2) Enhanced KNN: The classical KNN model has largely been applied to different kind of problems other than traffic flow prediction. The KNN framework commonly used is the one described above. However, several improvements can be introduced based on the problem's nature. In our problem, various attributes can be regarded in order to boost forecast

accuracy. Therefore to enhance the performance of our KNN model, we incorporate the following characteristics as well:

- Detector-wise: although the flow differs from one detector to another as shown in Figure 1, it may happen that patterns from different detectors have a partial similarity. Since these candidates are retrieved from different detectors, the other part of them may be very different, which badly impacts the forecasting accuracy. Thus, if the prediction is to be made for a given detector, the model inspects data related to only this sensor.
- Weekday-wise: when our model explores the historical data to retrieve state vector profiles, it only considers the same weekday. For instance, if the current state vector is taken from a Wednesday, all the profiles are constructed from historical data belonging to Wednesdays. Observations showed that there is a clear difference between working-day's and weekend's flows. More precisely, even days of the same category differ in flow patterns, which justifies our choice. This weekday-wise pre-selection of state vector profiles showed a significant improvement in model's performance through preliminary experiments.
- State vector length: length of state vectors, denoted by l, indicates how far backwards from a given instant t the data is relevant to make accurate predictions. Hence, a state vector of length l is given by:

$$v(t-l,t) = [f(t-l), f(t-l+1), \ldots, f(t-1), f(t)] \quad (7)$$

  The length of state vector impacts the prediction quality as well. If l is relatively small, the information provided by the state vector may be insufficient to make accurate predictions. However, longer state vector might also provide irrelevant information. To choose the best value for l, preliminary experiments have been launched with different values of l, such that $l \in \{20, 30, 40, 50, 60, 90, 120, 150\}$. Tests showed that $l =$ 60 minutes (6 timestamps) is the best choice for our dataset.
- Search radius: To ensure that state vector profiles share similar characteristics with the current state vector, we only consider profiles within a certain radius denoted by R. This means that the model selects profiles falling no further than r timestamps forwards and backwards from a current instant t. Therefore, the search space is constrained within $t - r$ and $t + r$. Obviously as we decrease R, the ratio of profiles closer to the current state vector in terms of characteristics increases, and vice versa. One issue can be raised here, that is when R getting smaller, profiles get fewer, which may also affect prediction accuracy. Consequently, a trade-off value of R has to be determined in this respect. Experiments included $R \in \{40, 50, 60, 90, 120, 150, 200, 300\}$ showed that $R = 90$ minutes ($r = 9$ timestamps) is the best search radius for our experiments. Note that above 200 minutes (20 timestamps), the efficiency drastically decreases, which indicates that search radius imposition is worthy.

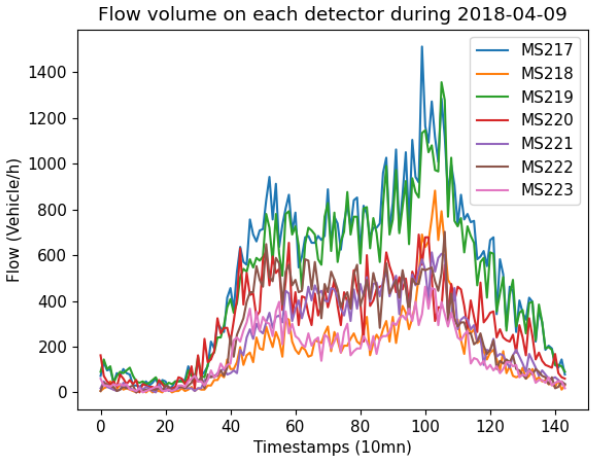


Fig. 1: Flow on detectors MS217-MS223

III. Data description

The work done in this paper is part of a project currently conducted at Center for Industrial Mathematics (ZeTeM) at the University of Bremen. The aim of the project is to model traffic in the city of Bremen in order to make accurate forecasts of different characteristics of traffic, especially traffic flow. The ultimate goal of the project is to set the stage for decision makers to take actions, in the context of fighting against climate change and air pollution, helping to reduce CO_2 emissions due to traffic. In this project we have large datasets of around 4 years worth of data. The data is gathered from over 550 measurement sites all around the city, on each of which an inductive loop detector is installed. This data is mainly delivered from the Traffic Management Center of Bremen (VMZ), which is an associated partner in our project. Figure 2 displays, in red bullets, the location of the set of loop detectors installed all around the city of Bremen to record traffic attributes. We focus in this paper on a junction located in city center surrounded by 7 loop detectors (MS217-MS223), as shown in Figure 3. This junction is situated in front of the main train station, tram station and bus station, which makes the traffic in this part very messy and subject to a lot of factors. Traffic lights are highly present in this area, but unfortunately we have no data about them. As in any data gathering device, due to malfunctioning, repairing or to data transmission, a lot of entries are missing in the final output recorded in databases. For this reason we selected a time-frame where we have almost complete data (98%) to be used for model assessment. Detectors take measurements each 90s, however in our study we use 10-minutes accumulations. The chosen time-frame is ranged from 09 April 2018 to 24 June 2018, which covers a period of 11 weeks. In order to measure the accuracy of the forecasts delivered by E-KNN

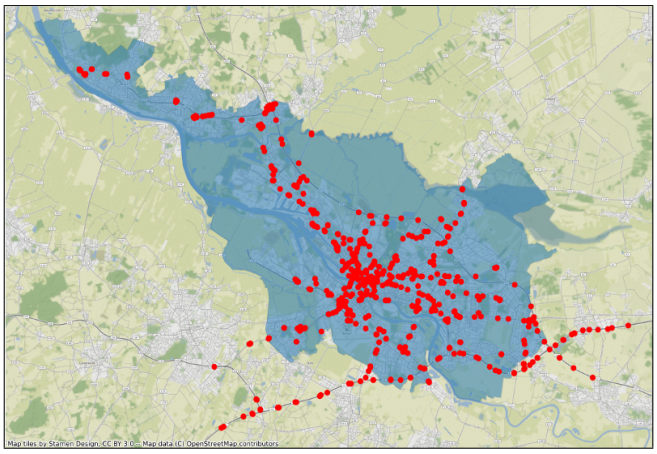


Fig. 2: Location of the detectors (in red bullets) installed all over Bremen.

model, we divide our dataset into two parts. The first consists of 8 weeks used for training, followed by 3 weeks for testing. More precisely, training takes place from 09 April 2018 to 03 June 2018, then we test for the period going from 04 June 2018 to 24 June 2018.

Fig. 3: Detectors location in the studied junction.

IV. Experimental results

We report and comment in this section the performance of E-KNN model under the data-set described in the previous section. The discussion of the output of E-KNN will be carried out according to the Mean Average Error (MAE) and Mean Absolute Average Error (MAPE) given in the formulas below:

$$MAPE = \frac{100}{n} \cdot \sum_{i=1}^{n} \left| \frac{m(t)_i - p(t)_i}{m(t)_i} \right| \tag{8}$$

$$MAE = \frac{1}{n} \cdot \sum_{i=1}^{n} |m(t)_i - p(t)_i| \tag{9}$$

such that $m(t)$ is the real value of traffic flow measured at instant t and $p(t)$ is the value predicted by the model. n is the number of predictions.

We launch E-KNN model to run on each detector separately because the data is different from one detector to another as mentioned earlier (refer to Figure 1). This is due to the detector's location, since some of them are operating on 1-lane roads, whereas others record data from 2-lane roads. Thus, the order of magnitude of flow volume is different on each detector. Predictions are made for 6-steps at once, meaning that we get forecasts of the next hour per 10-minutes. As structured in Table I, we report the performance accuracy based on day-hours. Therefore, the following categories are considered: traffic flow for all-day-hours, significant traffic hours (between 6h in the morning and 22h in the evening), morning peak times (from 6h to 9h) and finally evening peak times (from 16h to 19h). It is obvious that the results vary from one detector to another, and this is related, firstly, to the same reasons mentioned above (distinct flow); and secondly to traffic lights, which we have no data about them. Therefore, the accuracy of E-KNN is henceforth discussed by average results in each category of day-hours. First of all, one can notice that when the flow volume is significant, the MAE tends to grow while the MAPE tends to decrease. Since the flow volume in our data is relatively large, which can reach more than 1400 vehicle/hour (see Figure 1), we can expect this kind of somehow large MAEs. Note that the results have as well been affected by the occasional outliers, which we did not remove from our dataset. Taking into account all-day-hours, E-KNN model makes forecasts with an error of 46.40 MAE and 26.38% MAPE (random samples are exhibited in Figure 4). However, when we only regard significant flow during the day, meaning traffic in between 06:00 and 22:00, a worse MAE is reached (56.26) and a better MAPE is given (17.91%). It is expected to have a worse MAE because during this period of time larger numbers of vehicles are flowing. The accuracy in this category of hours is satisfactory as it reaches around 83% accuracy. Moreover, we also have an overview on the performance of the model during peak times in the morning and in the evening, in particular, from 06:00 to 09:00 and from 16:00 to 19:00. Absolute relative error in the morning is around 22% whereas it is only 16% in the evening. Given that the detectors are installed on signalized and heterogeneous urban arterial roads, which causes sharp fluctuations in the traffic, and we have no data on how traffic lights are programmed, we think that the accuracy of E-KNN model is satisfactory especially during rush day-hours.

V. Conclusion

An enhanced k-Nearest Neighbor technique has been introduced in this paper to deal with the problem of traffic flow volume forecast. The proposed model is applied to a non-processed dataset taken from urban roads located in downtown of Bremen (Germany). The model was trained with 8 weeks of data and tested with 3 weeks of it. We used MAE and MAPE as performance criteria, and measured the accuracy of E-KNN with regard to different categories of day-hours. When traffic is

TABLE I: 6-steps prediction results over the test-set.

	06:00-09:00		16:00-19:00		06:00-22:00		All-day	
Detector ID	MAE	MAPE	MAE	MAPE	MAE	MAPE	MAE	MAPE
MS217	63.58	14.30	100.50	12.10	75.08	11.97	62.87	16.64
MS218	23.96	28.86	55.94	20.29	36.11	22.78	28.80	30.57
MS219	61.54	15.34	92.90	11.98	71.28	12.35	60.41	18.80
MS220	73.98	21.12	80.82	17.16	74.10	18.30	62.32	27.12
MS221	26.24	25.25	53.18	16.99	45.87	21.58	38.24	31.30
MS222	38.42	26.63	57.00	17.77	47.59	19.31	37.11	30.54
MS223	47.54	23.98	49.78	17.45	43.80	19.09	35.06	29.68
Average	47.89	22.21	70.02	16.25	56.26	17.91	46.40	26.38

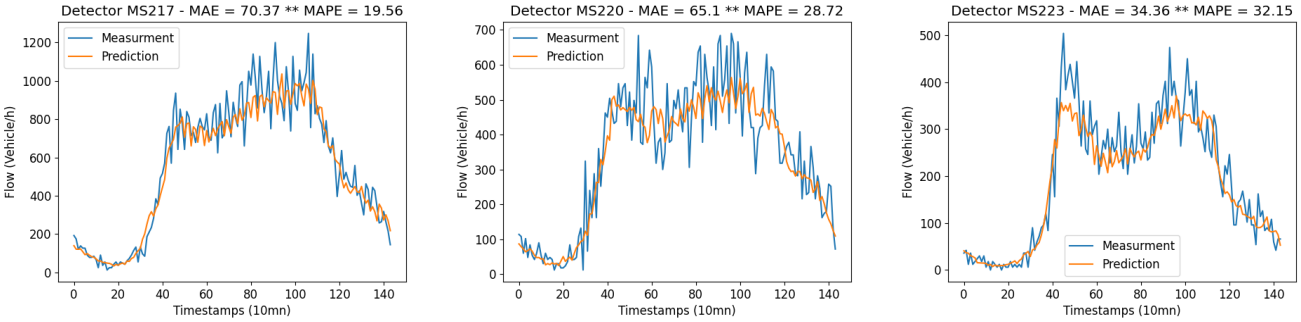


Fig. 4: Random samples of one-day prediction from different detectors.

significant during the day (between 06h and 22h), our model reaches around 83% accuracy, which can be considered as satisfactory given the nature of our dataset. Currently, we are working on the imputation of missing entries using different techniques to obtain complete data. Consequently, we would like to measure the performance of E-KNN model with larger, processed and filled in datasets.

REFERENCES

[1] Lei Bai, Lina Yao, Can Li, Xianzhi Wang, and Can Wang. Adaptive graph convolutional recurrent network for traffic forecasting. *Advances in Neural Information Processing Systems*, 33, 2020.

[2] Pinlong Cai, Yunpeng Wang, Guangquan Lu, Peng Chen, Chuan Ding, and Jianping Sun. A spatiotemporal correlative k-nearest neighbor model for short-term traffic multistep forecasting. *Transportation Research Part C: Emerging Technologies*, 62:21–34, 2016.

[3] Manoel Castro-Neto, Young-Seon Jeong, Myong-Kee Jeong, and Lee D Han. Online-svr for short-term traffic flow prediction under typical and atypical traffic conditions. *Expert systems with applications*, 36(3):6164–6173, 2009.

[4] Shifen Cheng, Feng Lu, Peng Peng, and Sheng Wu. Short-term traffic forecasting: an adaptive st-knn model that considers spatial heterogeneity. *Computers, Environment and Urban Systems*, 71:186–198, 2018.

[5] Gary A Davis and Nancy L Nihan. Nonparametric regression and short-term freeway traffic forecasting. *Journal of Transportation Engineering*, 117(2):178–188, 1991.

[6] Peibo Duan, Guoqiang Mao, Changsheng Zhang, and Shangbo Wang. Starima-based traffic prediction with time-varying lags. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1610–1615. IEEE, 2016.

[7] Filmon G Habtemichael, Mecit Cetin, and Khairul A Anuar. Methodology for quantifying incident-induced delays on freeways by grouping similar traffic patterns. In *Transportation Research Board 94th Annual Meeting*, pages 15–4824, 2015.

[8] Daniel Klosa, Amin Mallek, and Christof Büskens. Short-term traffic flow forecast using regression analysis and graph convolutional neural networks. *2021 IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, pages 1413–1418, 2021.

[9] S Vasantha Kumar and Lelitha Vanajakshi. Short-term traffic flow prediction using seasonal arima model with limited input data. *European Transport Research Review*, 7(3):1–9, 2015.

[10] Longshun Liu. A short-term traffic flow prediction method based on svr. In *2021 2nd International Conference on Urban Engineering and Management Science (ICUEMS)*, pages 1–4. IEEE, 2021.

[11] Saiqun Lu, Qiyan Zhang, Guangsen Chen, and Dewen Seng. A combined method for short-term traffic flow prediction based on recurrent neural network. *Alexandria Engineering Journal*, 60(1):87–94, 2021.

[12] Yisheng Lv, Y. Duan, Wenwen Kang, Zhengxi Li, and F. Wang. Traffic flow prediction with big data: A deep learning approach. *IEEE Transactions on Intelligent Transportation Systems*, 16:865–873, 2015.

[13] Abolghasem Sadeghi-Niaraki, Parima Mirshafiei, Maryam Shakeri, and Soo-Mi Choi. Short-term traffic flow prediction using the modified elman recurrent neural network optimized through a genetic algorithm. *IEEE Access*, 8:217526–217540, 2020.

[14] Brian L Smith and Michael J Demetsky. Traffic flow forecasting: comparison of modeling approaches. *Journal of transportation engineering*, 123(4):261–266, 1997.

[15] Brian L Smith, Billy M Williams, and R Keith Oswald. Comparison of parametric and nonparametric models for traffic flow forecasting. *Transportation Research Part C: Emerging Technologies*, 10(4):303–321, 2002.

[16] Mascha Van Der Voort, Mark Dougherty, and Susan Watson. Combining kohonen maps with arima time series models to forecast traffic flow. *Transportation Research Part C: Emerging Technologies*, 4(5):307–318, 1996.

[17] Zuduo Zheng and Dongcai Su. Short-term traffic volume forecasting: A k-nearest neighbor approach enhanced by constrained linearly sewing principle component algorithm. *Transportation Research Part C: Emerging Technologies*, 43:143–157, 2014.

sustainability

MDPI

Article

Impact of Data Loss on Multi-Step Forecast of Traffic Flow in Urban Roads Using K-Nearest Neighbors

Amin Mallek *, Daniel Klosa and Christof Büskens

WG Optimisation and Optimal Control, Center for Industrial Mathematics, University of Bremen, 28359 Bremen, Germany
* Correspondence: amallek@uni-bremen.de

Abstract: Data-driven models have recently proved to be a very powerful tool to extract relevant information from different kinds of datasets. However, datasets are often subject to multiple anomalies, including the loss of important parts of entries. In the context of intelligent transportation, we examine in this paper the impact of data loss on the behavior of one of the frequently used approaches to address this kind of problems in the literature, namely, the k-nearest neighbors model. The method designed herein is set to perform multi-step traffic flow forecasts in urban roads. In our study, we deploy non-prepossessed real data recorded by seven inductive loop detectors and delivered by the Traffic Management Center (VMZ) of Bremen (Germany). Firstly, we measure the performance of the model on a complete dataset of 11 weeks. The same dataset is then used to artificially create 50 incomplete datasets with different gap sizes and completeness levels. Afterwards, in order to reconstruct these datasets, we propose three computationally-low techniques, which proved through empirical testing to be efficient in reproducing missing entries. Thereafter, the performance of the E-KNN model is assessed under the original dataset, incomplete and filled-in datasets. Although the accuracy of E-KNN under incomplete and reconstructed datasets depends on gap lengths and completeness levels, under original dataset, the model proves to deliver six-step forecasts with an accuracy of 83% on average over 3 weeks of the test set, which also translates to a less than one car per minute error.

Keywords: data loss; incomplete dataset; intelligent transportation; k-nearest neighbors; linear regression; short-term forecast; traffic flow

Citation: Mallek, A.; Klosa, D.; Büskens, C. Impact of Data Loss on Multi-Step Forecast of Traffic Flow in Urban Roads Using K-Nearest Neighbors. *Sustainability* **2022**, *14*, 11232. https://doi.org/10.3390/su141811232

Academic Editor: Armando Cartenì

Received: 8 August 2022
Accepted: 2 September 2022
Published: 7 September 2022

1. Introduction

A huge part of business and economy nowadays heavily relies on transportation systems. Among others is e-commerce, which is partly based on delivering goods to customers, and transportation of individuals from and to work, for example. Optimizing costs and time for such operations requires efficient and intelligent transportation management systems. One of the crucial components of these systems is the prediction of different attributes related to traffic, especially traffic flow volume. Often, this latter is the main element on which other traffic-related features are based, and is a vital topic discussed in both academia and industry. Traffic managers usually depend on short-term traffic flow forecasts to plan and formulate efficient strategies in order to alleviate road congestion and further optimize vehicular traffic inside cities. Moreover, travelers also refer to these forecasts to take decisions about their traveling plans. The development of approaches for the purpose of accurate short-term flow forecasting might not be successful without a large amount of data. Therefore, traffic management centers deploy a large range of tools to monitor and record traffic attributes, including inductive loop detectors, video and image processing, radars of different kinds, and other Internet of Things (IoT) mechanisms. Recently, and in the past years, researchers have extensively examined the problem of short-term traffic flow forecasts. Consequently, several data-driven models have been proposed to tackle

this problem, categorized into two main classes: parametric and non-parametric models. We can briefly describe parametric models as the models that output predictions based on an explicit function defined within a finite set of parameters. These parameters are often estimated by training the model with a given dataset, for instance: ARIMA and its variants [1–3], neural networks [4–6], deep learning [7], and linear regression [8]. In contrast, non-parametric methods deliver predictions without assuming any prior knowledge or having any explicit formulas, such as support vector regression (SVR) [9,10] and k-nearest neighbors [11–13].

In the literature, researchers generally consider datasets with a few missing entries that are usually imputed using simple techniques, otherwise, they take into account only valid data. However, only a few of them drew attention to the impact of data loss on the performance of predictive models. Therefore, some techniques have been used in the literature to substitute corrupted and missing entries with valid data. First attempts were made, for instance, by Nihan et al. [14] by deploying the classical auto-regressive integrated moving average model. Zhong et al. [15] proposed different techniques to substitute missing input, including neural networks and regression models. Later on, a non-parametric spatio-temporal kernel regression model is developed to forecast travel time under the assumption of sensor malfunction. The results were compared to a k-nearest neighbors model, which is also non-parametric. The k-nearest neighbors technique has also been used for traffic data imputation in [16]. Tian et al. proposed a long short-term memory-based neural network that efficiently circumvents the negative impact of data loss [17]. Duan et al. [18] employed a deep learning-based approach called denoising stacked autoencoders for efficient imputation of missing data. Teresa Pamula [19] investigated the sensitivity of neural networks to loss of data in traffic flow prediction and proposed a strategy to substitute lost data in a way where the accuracy of forecasts is maintained. Some statistical models, including Markov chains, PPCA-based approaches, and Monte Carlo simulations have also been used [20–23]. An automated imputation procedure based on an adaptive identification technique that tries to minimize the error between simulated and measured densities was elaborated by Muralidharan and Horowitz [24]. Other techniques such as replacement by null values, substituting by the sample mean, or exponentially moving average were considered for testing and they showed good performance practically [25,26]. Several other strategies have as well been used including fuzzy C-means hybridized with a genetic algorithm [27], tensor-based methods [28], and simulator software such as Sumo and TransWorld [29].

In this paper, we address the problem of multi-step flow volume forecast in urban roads under the circumstances of data loss. This problem is part of the DiSCO2 project conducted at the University of Bremen. The aim of this project is to set the stage for decision makers to take actions to reduce CO_2 emissions in Bremen (Germany). To this end, we develop an enhanced k-nearest neighbors model based on traffic features. This method (shortened as KNN), is known to be a non-parametric data-driven model and has extensively been investigated in the literature. Old KNN models mainly focus on single-step forecasts [30–32]. However, this technique and other non-parametric models have the advantage of being flexible and easily extensible. Therefore, herein we extend the classical k-nearest neighbors to what we call enhanced KNN, referred to as the E-KNN model. This latter takes into account more attributes related to traffic to improve forecasting accuracy. Several improvements to the KNN model have already been considered in the context of traffic flow in some papers, including [12], where the authors deployed a weighted Gaussian method to compute forecasts instead of the typical ones, as in [31]. The authors in [11] incorporated a time constraint in neighbor selection and a minima distance to avoid the selection of highly auto-correlated candidates. Cheng et al. [13] developed a KNN model based on the assumption that traffic between adjacent road segments within assigned time periods is not correlated. This spatio-temporal approach comprehensively considers the spatial heterogeneity of traffic. Our E-KNN takes into account a search radius to ensure that selected profiles share similar characteristics. It also assumes that the flow is not only distinct between weekends and working days, but also among all

weekdays. Usually, studies are carried out on processed, filtered, or normalized data, herein we measure the performance of our designed technique using raw data, provided by the Traffic Management Center (VMZ) of Bremen, with no preprocessing, meaning that noise, corrupted data, and outliers are kept as they are in our dataset. The purpose behind this is to have an idea about the accuracy of the model when it is operated online, as needed in our project. Furthermore, for the same reason, the model performs six-step (1 h) forecasts at once in order to reduce the computational time.

In the second part of the paper, we take out the same dataset used to measure the performance of E-KNN and create artificially incomplete datasets. To do so, we try to simulate the actual status of most raw datasets (including ours). Thus, we produce 50 datasets with different gap sizes and completeness levels. Afterward, we try to reconstruct the missing parts of these datasets by deploying three different techniques that we designed for this purpose. We first assess the accuracy of reconstructing these datasets and profoundly examine their structure, then apply E-KNN to each of them. At this point, we can obtain an overview of how the E-KNN model behaves when is applied to incomplete and partially reconstructed datasets. A deep analysis of this latter is thoroughly reported afterward.

The rest of this paper is structured as follows. In Section 2 we describe the basic framework of k-nearest neighbors, then introduce the enhanced version of this model, referred to as E-KNN. Section 3 comprises the imputation techniques designed to fill in incomplete datasets. An in-depth description of the dataset used in this paper, and further in some parts of our project, is sketched in Section 4. The way the incomplete datasets are created and reconstructed is also extensively reported in the same section. Afterward, we detail in Section 5 the empirical findings out of testing the performance accuracy of E-KNN under original as well as incomplete and filled-in datasets. Finally, the paper is concluded in Section 6.

2. K-Nearest Neighbors Model

The k-nearest neighbors model is one of the famous data-driven predictive models applied to different problems present in the literature. In its essence, this approach explores historical data to fetch patterns in the past similar to a present one. The algorithm then tries to generate forecasts based on the future states of the historical patterns that are hypothetically closely similar to a current state. The quality of this model as any other data-driven model depends on how big the data is and also how well it is represented. In what follows, we detail the basic components of KNN and how they are characterized. Afterward, based on the nature of our problem, we integrate into the model some enhancements to improve its performance.

2.1. Basic KNN and Notations

In this subsection, we introduce the set of notations used to describe our KNN model and its functionalities. First of all, traffic flow volume at instant t at some detector (sensor) d is denoted by $f_d(t)$. Thus, a flow volume series at some detector d in a given time frame between t_i and t_j $(i \leq j)$ is defined by the following vector:

$$v_d(t_i, t_j) = [f_d(t_i), f_d(t_{i+1}), \ldots, f_d(t_{j-1}), f_d(t_j)] \tag{1}$$

For simplicity, we concisely write $f(t)$ and $v(t_i, t_j)$, unless the detector is required to be mentioned. We define a state vector of length l at instant t as a vector of flow volume comprising the traffic flow from instant t backward to instant $t - l$, therefore it can be seen as:

$$v(t-l, t) = [f(t-l), f(t-l+1), \ldots, f(t-1), f(t)] \tag{2}$$

The forecasts of the next s steps are given in a prediction vector denoted by v' and expressed as follows:

$$v'(t+1, t+s) = [f'(t+1), f'(t+2), \ldots, f'(t+s-1), f'(t+s)] \tag{3}$$

Once the state vector is defined, the other components of the KNN framework can then be set. The first task is to select a set of nearest neighbors to a given state vector based on measuring a certain distance between this latter and other candidate vectors. The best k candidates (neighbors) are then selected to be considered in the prediction process. Various methods are usually used to determine the closeness of the current state vector to other vectors or to produce forecasts. The distance between two state vectors is commonly given by the Euclidean distance, however sometimes and in some cases, the correlation coefficient distance is also considered where its superiority is proven [11]. Averaging the k nearest neighbors is often the common way used to predict future states. Though, many other approaches were also applied in the literature to obtain forecasts, including weighting the k nearest neighbors according to their distance to current state vector [33] and Gaussian-weight distance [12,13].

Our KNN algorithm is designed to perform multi-step prediction, therefore, at some instant t, the vector aggregating current state, and s future steps can be seen as follows:

$$E(t) = v(t-l,t) + v'(t+1,t+s) = [f(t-l),\ldots,f(t),f'(t+1),\ldots,f'(t+s)] \quad (4)$$

In our study, we use the Euclidean distance to rank the neighbors of state vectors. Firstly, the correlation coefficient distance proved to be inferior to the Euclidean distance through the experiments. Secondly, the Gaussian-based distance did not increase the accuracy of the predictions. We also use simple averaging of the k nearest neighbors. Note that weighting of neighbors procedure has also been tested and gave the same results as simple averaging. Mathematically, the Euclidean distance, in our case, is given as follows:

$$dist^{(i,j)}(v^i,v^j) = \sum_{\lambda=0}^{l} (f^i(t-\lambda) - f^j(t-\lambda))^2 \quad (5)$$

Such that v^i and v^j are two state vectors. As said before, our KNN is designed to forecast multiple steps, hence the formula to deliver a prediction vector is:

$$v'(t+1,t+s) = [f'(t+1),\ldots,f'(t+s)] = [\sum_{i=1}^{k} f^{(i)}(t+1)/k,\ldots,\sum_{i=1}^{k} f^{(i)}(t+s)/k] \quad (6)$$

2.2. Enhanced KNN

The classical KNN model has extensively been applied to different kinds of problems other than traffic flow prediction. The KNN framework commonly used is the one just described. However, several improvements can be introduced based on the problem's nature. In our problem, various features can be regarded in order to boost forecast accuracy. Therefore to enhance the performance of our KNN model, henceforth referred to as E-KNN, we incorporate the following characteristics as well:

- Detector-wise: although the flow differs from one detector to another as shown in Figure 1, it may happen that patterns from different detectors have a partial similarity. Since these candidates are retrieved from different detectors, the other parts of them may be very different, which badly impacts the forecasting accuracy. Thus, if the prediction is to be made for a given detector, the model inspects only data related to it.
- Weekday-wise: when our model explores the historical data to retrieve state vector profiles, it only considers the same weekday. For instance, if the current state vector is taken from a Wednesday, all the profiles are constructed from historical data belonging to Wednesdays. Observations showed that there is a clear difference between working-day and weekend flows. More precisely, even days of the same category differ in flow patterns, which justifies our choice. This weekday-wise pre-selection of state vector profiles showed a significant improvement in the model's performance through preliminary experiments.

- State vector length: the length of state vectors, denoted by l, indicates how far backward from a given instant t the data is relevant to make accurate predictions. Hence, a state vector of length l is given by:

$$v(t-l,t) = [f(t-l), f(t-l+1), \ldots, f(t-1), f(t)] \quad (7)$$

  The length of state vectors impacts the prediction quality as well. If l is relatively small, the information provided by the state vector may be insufficient to make accurate predictions. However, a longer state vector might also provide irrelevant information. To choose the best value for l, preliminary experiments have been launched with different values of l, such that $l \in \{20, 30, 40, 50, 60, 90, 120, 150\}$. Tests showed that $l = 60$ min (six timestamps) is the best choice for our dataset.
- Search radius: to ensure that state vector profiles share similar characteristics with the current state vector, we only consider profiles within a certain radius denoted by R. This means that the model selects profiles falling no further than r timestamps forwards and backwards from a current instant t. Therefore, the search space is constrained within $t - r$ and $t + r$. Obviously, as we decrease R, the ratio of profiles closer to the current state vector in terms of characteristics increases, and vice versa. One issue can be raised here, when R becomes smaller, profiles become fewer, which may also affect the prediction accuracy. Consequently, a trade-off value of R has to be determined in this respect. Experiments included $R \in \{40, 50, 60, 90, 120, 150, 200, 300\}$ and showed that $R = 90$ min ($r = 9$ timestamps) is the best search radius for our experiments. Note that above 200 min (20 timestamps), the efficiency drastically decreases, which indicates that search radius imposition is worthy.

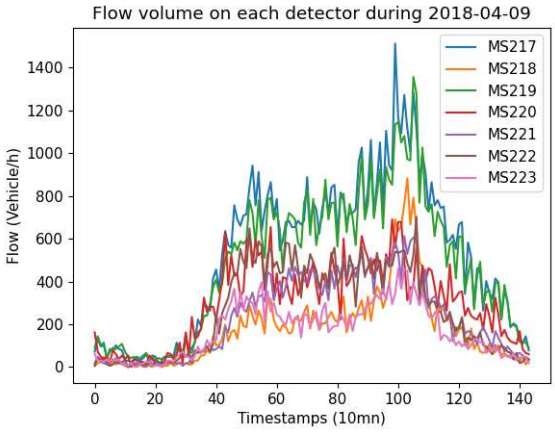


Figure 1. Flow volume on detectors MS217–MS223 on 9 April 2018.

3. Imputation Techniques

Any kind of time series generally contains a mix of invalid values, missing entries, and outliers; flow time series are not an exception. Commonly, missing and corrupted values are replaced by some kind of constant depending on the predictive model to be used, however, usually outliers are replaced with the closest rational value. In this section we introduce three different techniques with a low computational effort to be deployed for imputing missing values in our time series. The most efficient strategy among them will be used to impute missing entries in our project. The reason behind seeking low computational effort is that the chosen method is to be integrated into an online system.

3.1. Mean

An intuitive approach is to compute the mean of each timestamp over the entire dataset. This roughly gives the amount of flow at each period of time during the day. Without loss of generality, for 10 min accumulation we have 144 timestamps per day, we denote their flow value by $f(t)$, $t \in \{1, 2, \ldots, 144\}$. Let us assume that in our dataset we

have D days and only ω available values of $f(t)$ (the rest is missing). Thus, we replace the missing flow values at timestamp t in our dataset by:

$$m(t) = \frac{\sum_{i=1}^{\omega} f(t)^{(i)}}{\omega} \tag{8}$$

Such that $f(t)^{(i)}$ is the i^{th} day with available flow at this timestamp (i^{th} available flow value at timestamp t). Note that this technique excludes any differences between weekday flows.

3.2. Mean per Weekday

This method is a more precise approach than the previous one. Herein, besides the mean, we take into account weekdays as well. This means we compute the mean for each timestamp of each weekday. If we take for instance 10 min accumulation, then we have to take into account 144×7 values. As mentioned above the flow differs from one weekday to another, especially during weekends. The idea is similar to the previous one, over the dataset, we compute the mean flow for each timestamp related to weekday $j \in \{1, \ldots, 7\}$. Consequently, the values can be given by the following formula:

$$m^{(j)}(t) = \frac{\sum_{i=1}^{\omega(j)} f(t)^{(i)}}{\omega(j)} \tag{9}$$

where $\omega(j)$ is the number of available values over the dataset at timestamp t of weekday j and $m^{(j)}(t)$ is the imputed value at timestamp t of weekday j.

3.3. Linear Regression

In this subsection, we briefly describe a regression analysis model that has been introduced for flow prediction in [8]. A linear regression model has been deployed, in which forecasts are given by a set of polynomials of different degrees. The general formula of these polynomials is:

$$P(x) = \sum_{k=0}^{n} \alpha_k x^k + \epsilon = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \ldots + \alpha_n x^n + \epsilon \tag{10}$$

where n is the degree of P and α is a vector of coefficients to be calibrated from the data. The term ϵ is a random error with mean zero added for bias.

The model uses regression per weekday and incorporates it with local regression, namely, hourly regression. First of all, the model tries to capture the regression of each weekday with a polynomial of degree 10. The hourly regression of the flow is learned per weekday as well with a polynomial of degree 5. As a result, the predicted values are given by combining both values (daily and hourly) with more emphasis on daily regression to avoid over-fitting. We will be using this model to predict the missing values at a given timestamp, then impute them accordingly.

4. Data and Reconstructed Data

The work done in this paper is part of the DiSCO2 project currently conducted at the Center for Industrial Mathematics (ZeTeM) at the University of Bremen. The aim of the project is to model the traffic in the city of Bremen in order to make accurate forecasts of different characteristics of traffic, especially traffic flow. The ultimate goal of the project is to set the stage for decision makers to take actions targeting the reduction of CO_2 emissions in the context of fighting against climate change and air pollution.

4.1. Data Description

In this project, we have large datasets of around 5 years' worth of data. The data is gathered from over 550 measurement sites all around the city, on each of which an inductive loop detector is installed. This data is mainly delivered by the Traffic Management Center of Bremen (VMZ), which is an associated partner in our project.

Figure 2 displays, in red bullets, the location of loop detectors installed all around Bremen to record traffic attributes. This paper is only concerned with one place of the city located in the city center. We focus on a junction surrounded by seven loop detectors (MS217–MS223), as shown in Figure 3. This junction is situated in front of the main train station as well as tram and bus stations, which makes the traffic in this area very messy and subject to a lot of factors. Traffic lights are highly present in this region, but unfortunately we have no data about them. As in any data gathering device, due to malfunctioning, repairing, or data transmission, many entries are missing in the final output recorded in databases. In our case, an important part of the data is missing over all 5 years. Sometimes values are missing for months, and further, the completeness level of many of the detectors is less than 50%. For this reason we selected a time frame where the data to be used is almost complete (98%). First, we will use this data to train and test the predictive model. Afterward, we will destroy parts of this dataset and then try to reconstruct it with the different imputation techniques mentioned in the previous section. Detectors take measurements each 90s, however in our study we use 10 min accumulations. The precise dates used are from 9 April 2018 to 24 June 2018, which covers a period of 11 weeks. As we already mentioned, this period corresponds to the time frame having the least amount of missing entries. In order to measure the performance of the imputation techniques and the accuracy of the forecasts delivered by the predictive model, we divide our dataset into two parts. The first one consists of 8 weeks used for training, followed by 3 weeks for testing. Precisely, training takes place from 9 April 2018 to 3 June 2018 then we test for the period going from 4 June 2018 to 24 June 2018. The best imputation strategy will be later used to fill in missing, corrupted, and outlier values in our database.

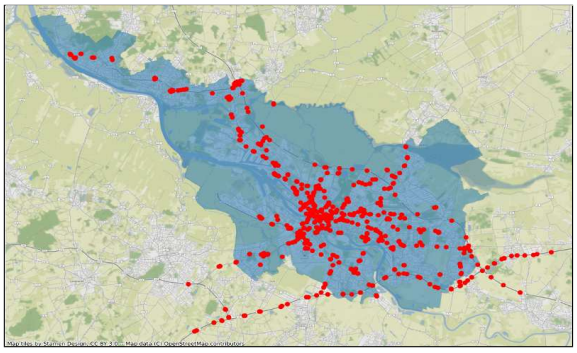


Figure 2. Location of the detectors (in red bullets) installed all over Bremen.

Figure 3. Detectors location in the studied junction.

4.2. Reconstructed Data

In this subsection, we describe how we destroy parts of the working dataset and reconstruct it. First, from the data described in the previous subsection, we take out the same time frame where we have almost complete data (98%). We then artificially create incomplete datasets by randomly removing different portions of data to reach a certain level of completeness. Since in our data we have different lengths of missing portions, ranging from one timestamp to even months, we will proceed by using analogous reasoning. We proceed by removing, at random, timestamp portions of one of the following sizes, $\{1, 3, 6, 36, 72, 144, 288, 1008, 4320\}$, respectively corresponding to 10 min, 30 min, 1 h, 6 h, 12 h, 1 day, 2 days, 1 week, and 1 month periods of time. The deletions are carried out at random points until we reach different levels of completeness: 50%, 60%, 70%, 80%, and 90%. We also construct incomplete datasets to reach the previous incompleteness ratios by passing in a list of random portion sizes, hence, there are different interval lengths of missing data in each dataset.

Via what we have just described, we create 50 variants of incomplete datasets having different combinations of gap lengths and incompleteness levels. To each of these we apply the imputation techniques reported in Section 3 to construct complete datasets.

4.3. Performance of Imputation Methods

In order to measure the performance of the imputation methods, we consider the same split mentioned above for our dataset. The first one is used to train the methods, and the second part is to test their performance. We use mean absolute error (MAE), given in Equation (12) as a criterion of accuracy. From the results reported in Tables 1–3 and their corresponding Figures 4–6, we clearly see that the three methods are closely competitive; however, it is obvious that the linear regression model is more accurate than the others. The performance of the three models varies in function of completeness level and gap lengths as well. Therefore, in what follows we comment and discuss the results based on these attributes.

- Completeness ratio: The experiments reported in Tables 1 and 2, respectively plotted in Figures 5 and 6, used different levels of completeness to investigate the impact of various missing portions of data. The results showed that the completeness percentage has an influence on the accuracy of the imputation methods. As we increase the number of missing entries, the performance quality of the three imputation methods decreases from around 91 with 50% completeness to 51 with 90% completeness. This is clearly apparent in Figure 6, where a list of random gap lengths is passed in. In contrast to that, Figure 5 shows that there is only a slight impact on the completeness ratio when deletions are based on fixed gap lengths. This kind of performance is mainly due to the large gaps of deletions (week and month), in this case deletions sometimes take place mostly in the training set and sometimes in the test set, which alternates the performance quality.
- Gap lengths: The results exhibited in Table 3 and Figure 4 suggest that for small gap deletions the performance of the models is worse than the one with larger gaps. When gap length is between 10 min and 1 day, the MAE is between 75 and 80, however, it drops down to around 72 for one week gap and 58 for one-month deletion. This kind of performance suggests, first, that the deletion of whole consecutive days has a smaller impact on the performance of the models than missing shorter entries for one day. Secondly, this means that training with smaller complete datasets is better than doing it with larger ones with multiple missing entries of a length less than one day. The efficiency of the models gets even better when the gap gets larger, namely one week and one month. In these cases, two possibilities are to be considered. The first is that the deletions are mostly (due to their length: a week or a month) in the training set, which means that only a few entries on the test set have to be imputed, which explains low errors (MAE). The other is that more missing entries are located in the

test set, thus the training set is somehow complete, which affected well the filling process of the missing values in the test set.

Table 1. Performance (MAE) of imputation methods in function of completeness ratio on fixed gap-lengths datasets.

Completeness Ratio	50%	60%	70%	80%	90%
Mean	77.28	78.71	76.24	79.11	79.09
Mean Weekday	78.26	75.80	74.04	79.11	76.46
Linear Regression	**76.95**	**74.82**	**73.26**	**78.42**	**76.01**

Table 2. Performance (MAE) of imputation methods in function of completeness ratio on datasets with a list of gap-lengths.

Completeness Ratio	50%	60%	70%	80%	90%
Mean	92.13	69.48	71.62	64.90	51.62
Mean Weekday	91.56	57.82	66.50	61.83	51.60
Linear Regression	**91.22**	**57.27**	**66.50**	**61.44**	**51.50**

Table 3. Performance (MAE) of imputation methods in function of gap-length.

Gap Length	Mean	Mean Weekday	Linear Regression
1	78.50	77.89	**76.85**
3	78.62	79.70	**78.33**
6	**78.33**	79.42	78.40
36	79.90	80.47	**79.65**
72	80.45	79.98	**78.85**
144	77.49	77.03	**76.39**
288	78.13	77.16	**76.56**
1008	76.51	72.55	**71.99**
4320	72.12	58.83	**58.10**

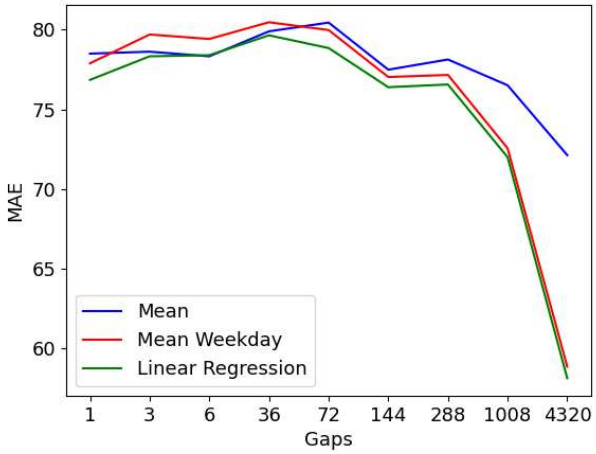


Figure 4. Performance of imputation methods in function of gap-length.

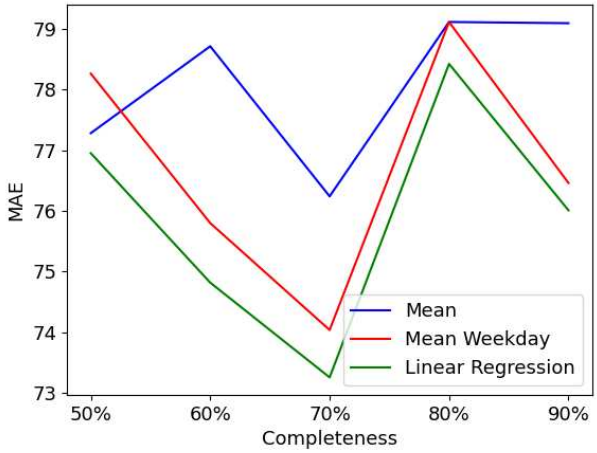


Figure 5. Performance of imputation methods for fixed gap-length.

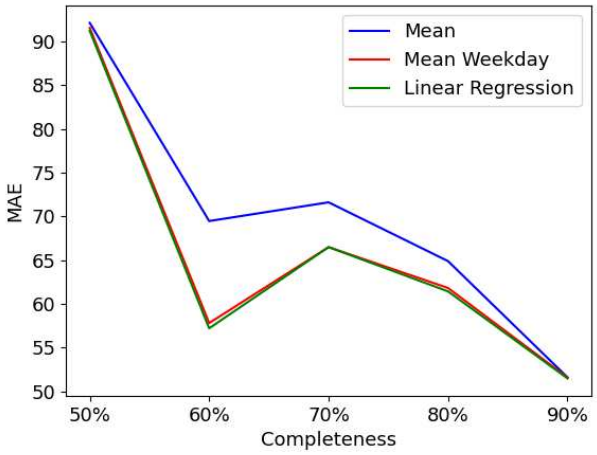


Figure 6. Performance of imputation methods for a list of gap-lengths.

4.4. Deviation between Original and Reconstructed Data

As introduced above, we artificially produced 50 datasets with multiple kinds of deletions, including fixed gap lengths and a list of random gaps under different percentages of completeness. Afterward, different models were applied to reconstruct missing entries in these datasets. This subsection quickly comments on some significant samples of distributions of original and reconstructed datasets. Distributions are plotted detector-wise, wherein the plots show the deviation between original, incomplete, and reconstructed datasets. The results are given in function of both completeness level and gap length, however, for brevity, herein we only include a few plots.

The plots in Figure 7 are taken from detector MS219 and aggregated by the percentage of completeness. We can clearly see that as we increase the percentage of incompleteness, the deviation between original and reconstructed datasets tends to grow, and vice versa. Although the filling methods are closely competitive, we can notice that linear regression has the least deviation from the original data, accordingly to what has been reported above. Similar conclusions can be drawn as we increase the gap length as well, as shown in Figures 8 and 9 taken from detector MS218. Note that this is also the case for almost all the other detectors.

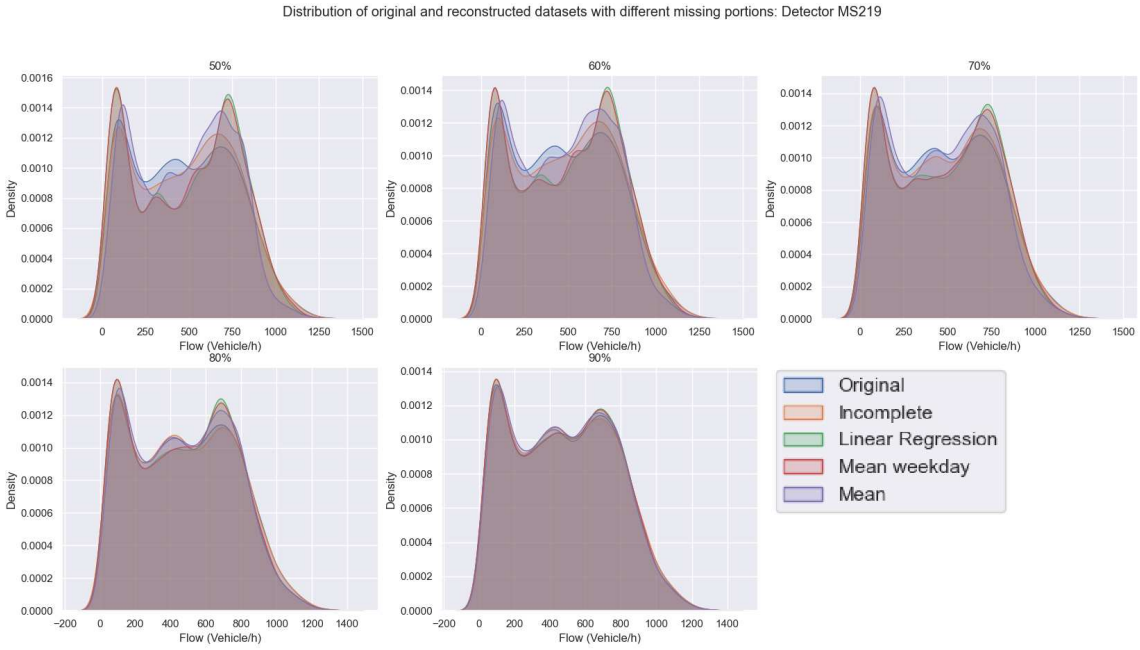


Figure 7. Deviation between original and reconstructed datasets with different missing portions: Detector MS219.

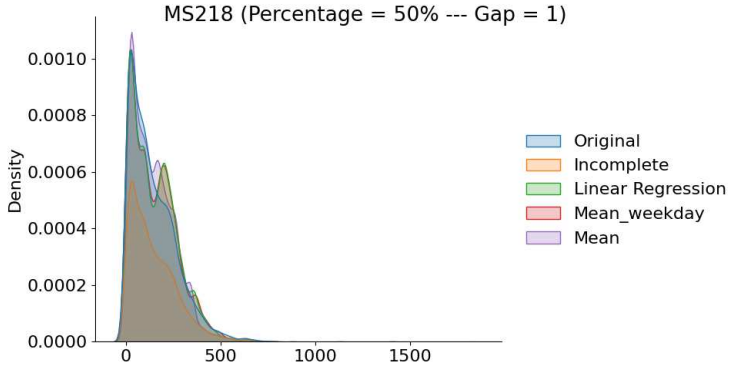


Figure 8. Distribution of original and reconstructed data with 50% completeness and gaps of length 1.

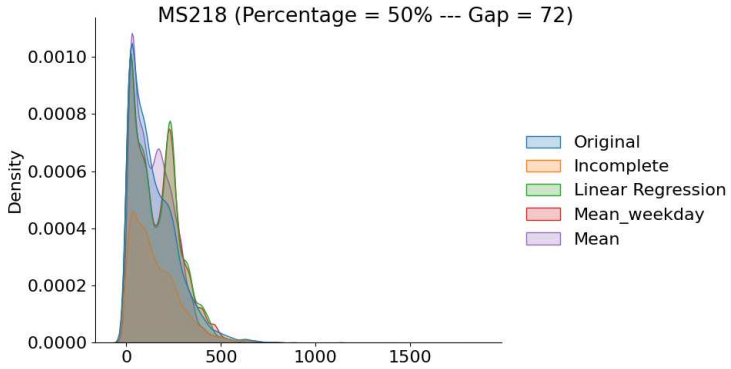


Figure 9. Distribution of original and reconstructed data with 50% completeness and gaps of length 72.

In order to give more insights into the deviation between original and reconstructed data, Figures 10–12 exhibit how the linear regression model reconstructs data. The figures are samples taken from the same day (9 April 2018) and different detectors (MS217, MS220,

and MS223). The data has 70% completeness level with missing entries of gap 1, 36, and 72 timestamps. We can notice that when we only have gaps of one timestamp missing at once, the data is somehow well reconstructed. As we increase the gap, the accuracy of the LR technique decreases. Figure 11 shows two gaps of 36 timestamps missing. We can see that the original data is very sparse, however, the LR model tries to reduce the effect of potential noise and outliers by imputing less sparse values. This is also set to avoid over-fitting as shown in Figure 12 as well.

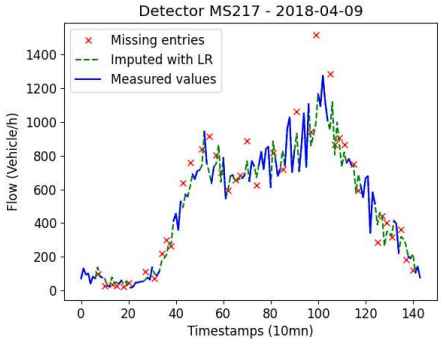


Figure 10. Data reconstructed with the Linear Regression method (completeness = 70%, gap = 1).

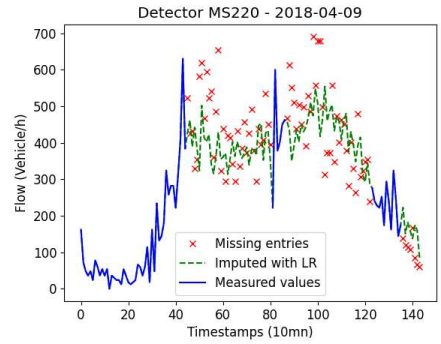


Figure 11. Data reconstructed with the Linear Regression method (completeness = 70%, gap = 36).

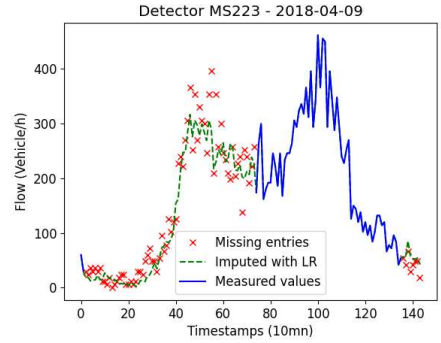


Figure 12. Data reconstructed with the Linear Regression method (completeness = 70%, gap = 72).

5. Results and Discussion

In this section, we report and discuss the output of our implemented model under both original and artificial datasets. Before doing so, in order to evaluate the accuracy of the predictions, two metrics are deployed: Mean Absolute Percentage Error (MAPE) and Mean Absolute Error (MAE). These are given by the following formulas:

$$MAPE = \frac{100}{n} \cdot \sum_{i=1}^{n} \left| \frac{m(t)_i - p(t)_i}{m(t)_i} \right| \quad (11)$$

$$MAE = \frac{1}{n} \cdot \sum_{i=1}^{n} |m(t)_i - p(t)_i| \quad (12)$$

Such that $m(t)$ is the real value of traffic flow measured at instant t and $p(t)$ is the value predicted by the model. n is the number of predictions.

5.1. Under Original Data

We report and comment in this subsection on the performance of the E-KNN model under the original dataset described in the previous section. We use the same dates for training and testing, namely, we train the model with data from 9 April 2018 to 3 June 2018, then test it on data from 4 June 2018 to 24 June 2018. The discussion of the output of E-KNN will be carried out according to the mean average error (MAE) and mean absolute average error (MAPE) given above (refer to Figures 13 and 14 for more details). We launch the E-KNN model to run on each detector separately because, as can be seen from Figure 1, the data is different from one detector to another. This is due to the location where the detectors are installed. Some loop detectors are collecting data from one-lane roads, whereas others record it from two-lane roads. Thus, the order of magnitude of the flow volume is different for each detector. Predictions are made for one hour, which means we forecast six steps at once. As structured in Table 4, we report the performance accuracy based on day hours. we consider traffic flow for all day hours, significant traffic hours (between 6 h in the morning and 22 h in the evening), morning peak times (from 6 h to 9 h), and finally evening peak times (from 16 h to 19 h). It is obvious that the results vary from one detector to another, and this is related, firstly, to the same reasons just mentioned (distinct flows), and secondly to traffic lights, which we have no data on. Therefore, the accuracy of E-KNN is henceforth discussed by averaging the results in each category. First of all, one can notice that when the flow volume is significant the MAE tends to grow while the MAPE tends to decrease. Since the flow volume in our data is relatively large, it can reach more than 1500 vehicles/hour (see Figure 1), we can expect this kind of somehow large MAEs. Note that the results have also been affected by the occasional outliers, which we did not remove from our dataset. Taking into account all day hours, the E-KNN model makes forecasts with an error of 46.40 MAE and 26.38% MAPE. However, when we only regard significant traffic during the day, meaning traffic between 06:00 and 22:00, a worse MAE is reached (56.26) and a better MAPE is delivered (17.91%). It is expected to have a worse MAE because during this period of time larger numbers of vehicles are flowing. The accuracy in this category of hours is satisfactory as it reaches around 83%. Moreover, two other categories appear as rush hours during the day, in particular, peak times in the morning and in the evening, from 06:00 to 09:00 and from 16:00 to 19:00. Relative absolute error in the morning is around 22%, whereas it is only 16% in the evening. Given that the detectors are installed on signalized urban arterial roads and we have no data on how traffic lights are programmed, we think that the accuracy of the E-KNN model is satisfactory, especially during rush hours.

Table 4. Prediction results over the test set for 1 h (6 steps).

	06:00–09:00		**16:00–19:00**		**06:00–22:00**		**All Day**	
Detector ID	**MAE**	**MAPE**	**MAE**	**MAPE**	**MAE**	**MAPE**	**MAE**	**MAPE**
MS217	63.58	14.30	100.50	12.10	75.08	11.97	62.87	16.64
MS218	23.96	28.86	55.94	20.29	36.11	22.78	28.80	30.57
MS219	61.54	15.34	92.90	11.98	71.28	12.35	60.41	18.80
MS220	73.98	21.12	80.82	17.16	74.10	18.30	62.32	27.12
MS221	26.24	25.25	53.18	16.99	45.87	21.58	38.24	31.30
MS222	38.42	26.63	57.00	17.77	47.59	19.31	37.11	30.54
MS223	47.54	23.98	49.78	17.45	43.80	19.09	35.06	29.68
Average	47.89	22.21	70.02	16.25	56.26	17.91	46.40	26.38

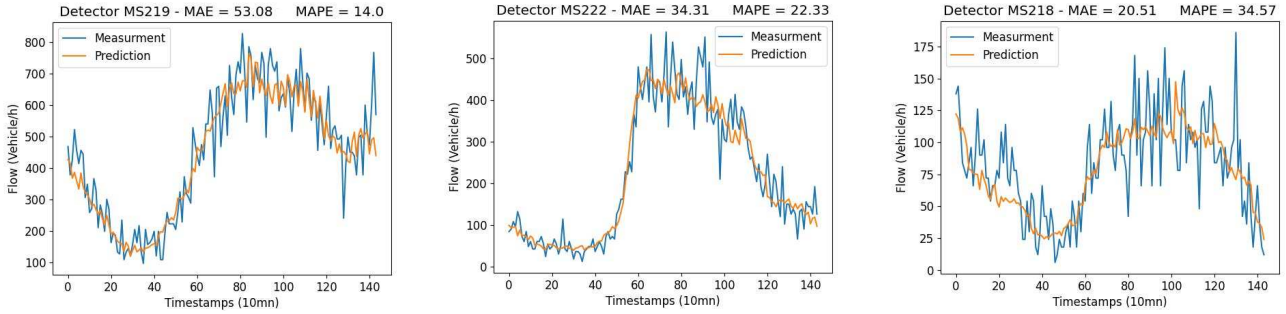


Figure 13. Predictions of traffic flow on randomly chosen detectors (weekend days).

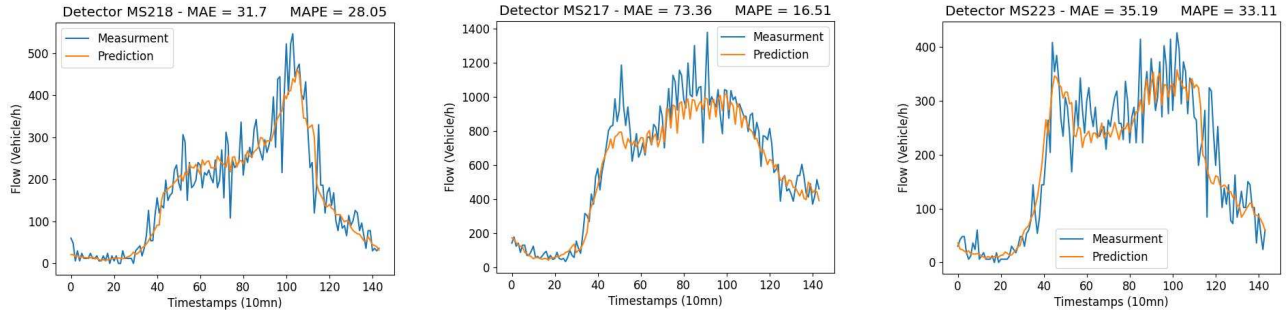


Figure 14. Predictions of traffic flow on randomly chosen detectors (working days).

5.2. Under Artificial Datasets

This subsection is dedicated to the discussion of E-KNN performance under the different imputed datasets produced as described in Section 4.2. In order to assess the performance of E-KNN under the filled-in datasets (50 datasets), we apply it to each of these datasets, always using the same time frames declared above for training and testing. The accuracy of the model is reported using the mean absolute error criterion (MAE) since both MAE and MAPE gave equivalent output. The results are discussed according to the level of completeness of each dataset and also following gap lengths. The results show that the ratio of completeness has an impact on the performance of E-KNN, however, not for all of the 50 datasets. This is also related to the positions of deletions (especially for large gaps) where sometimes they lay mostly in the training set and other times in the test set. Another factor that should be taken into account is the flow volume, which varies from one detector to another (refer to Figure 1) and sometimes leads to differences in errors when deletions take place mostly in either the training set or test set. From Tables 5 and 6, in general, as we increase the level of completeness of the datasets, E-KNN seems to perform better. The model's performance also depends on the length of gaps; for some of these configurations, the completeness level seems to have a huge impact on the performance ($gap = 36$, $gap = 1008$), and a slight one on others, as in $gap = 1$ for instance (see Figure 15). We think that this does not relate to the gap length itself but to the random distribution of deletions between the test set and training set in these datasets. Apart from that, completeness level has a huge impact, especially on incomplete datasets. For instance, on incomplete datasets with $gap = 144$, MAE decreases from 200 on a 60% level to around 80 on 90% completeness. Moreover, on datasets completed with the three models, MAE decreases from around 60 to 50 as we increase the completeness ratio. The same thing can be noticed with $gap = 36$ (see Figure 15), where for incomplete datasets MAE decreases from 60 to 55 and from around 65 to 47 for completed datasets. For $gap = 1008$ (see Figure 15), completeness level seems to have no effect on incomplete datasets, but intensely impacts the performance of E-KNN with filled-in datasets as MAE decreases from 65 to around 47.

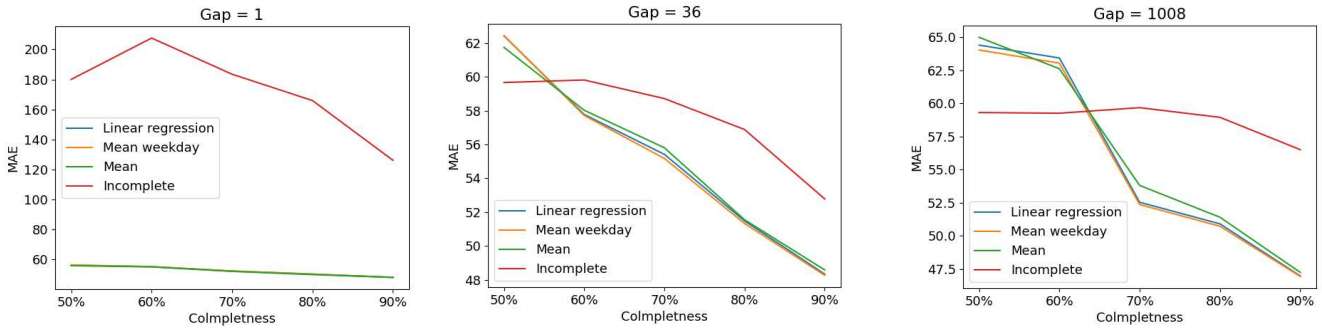


Figure 15. Performance of E-KNN under incomplete and completed datasets with different gap-lengths.

On the other side, the gap length parameter also has an impact on E-KNN performance. In general, when gap length is increased, the accuracy decreases (refer to Table 7). However, this is not the case for our results sorted in function of completeness level. It seems that, in presence of filled-in datasets, the gap length has little impact on the performance of E-KNN which means that the datasets are somehow efficiently completed regardless of the different deletion kinds. Howbeit, for incomplete datasets, gap length has an effect on the accuracy. Take for instance 90% completeness (refer to Figure 16), MAE increases as we increase the gap length, except for 72 and 288 where training sets seem to have fewer deletions than test sets, which allowed E-KNN to perform well. In this case, MAE jumps from around 50 on gap 1 to around 200 on gap 4320.

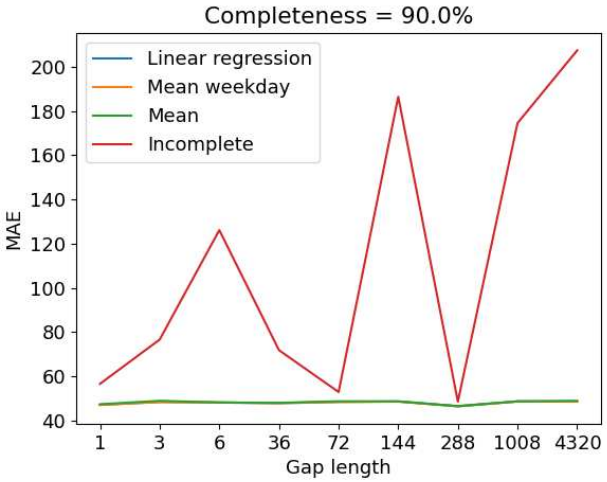


Figure 16. Performance of E-KNN on incomplete and completed datasets with a completeness level of 90%.

Table 5. E-KNN's performance for imputation methods in function of completeness ratio on datasets with a list of gap-lengths.

Completeness Ratio	**50%**		**60%**		**70%**		**80%**		**90%**	
	MAE	**MAPE**	**MAE**	**MAPE**	**MAE**	**MAPE**	**MAE**	**MAPE**	**MAE**	**MAPE**
Incomplete	157.19	158.73	150.03	127.85	102.47	114.57	120.49	112.31	58.73	37.23
Mean	**53.56**	32.32	54.09	34.57	53.10	33.00	51.16	30.14	46.97	27.93
Mean Weekday	53.76	29.24	**51.65**	29.45	**52.12**	29.81	**50.29**	27.39	**46.69**	26.35
Linear Regression	53.79	**29.10**	51.72	**29.38**	52.25	**29.72**	50.46	**27.21**	46.73	**26.30**

Table 6. E-KNN's performance for imputation methods in function of completeness ratio on datasets with fixed gap-lengths.

Completeness Ratio	50%		60%		70%		80%		90%	
	MAE	**MAPE**	**MAE**	**MAPE**	**MAE**	**MAPE**	**MAE**	**MAPE**	**MAE**	**MAPE**
Incomplete	124.91	111.74	167.87	139.26	159.17	159.63	142.12	131.63	111.20	93.35
Mean	60.35	38.45	56.99	35.66	53.69	33.49	50.79	30.55	48.09	28.01
Mean Weekday	60.15	33.34	**56.58**	31.61	**53.18**	29.95	50.63	28.40	**47.87**	26.83
Linear Regression	**60.09**	**33.17**	56.62	**31.48**	53.28	**29.85**	**50.27**	**28.34**	47.91	**26.78**

Table 7. E-KNN's performance in function of gap length.

Gap Length	Incomplete		Mean		Mean Weekday		Linear Regression	
	MAE	**MAPE**	**MAE**	**MAPE**	**MAE**	**MAPE**	**MAE**	**MAPE**
1	58.73	32.74	**52.16**	32.03	52.41	29.71	52.36	**29.58**
3	57.57	32.29	**53.22**	32.75	53.74	30.30	53.62	**30.16**
6	174.61	207.74	**53.34**	32.62	53.92	30.41	53.78	**30.27**
36	235.41	179.15	55.14	34.36	**54.98**	31.06	55.08	**30.98**
72	219.87	215.39	55.57	34.62	**54.89**	31.07	55.02	**31.01**
144	172.62	166.38	55.12	33.47	**54.71**	30.01	54.84	**29.95**
288	151.30	157.05	55.37	33.92	**54.11**	29.60	54.24	**29.51**
1008	144.13	139.71	56.00	35.25	**55.41**	30.79	55.64	**30.74**
4320	70.35	47.85	49.92	30.08	48.97	27.27	**48.95**	**27.12**

In a nutshell, based on the discussion above, we can say that the E-KNN model is very sensitive to incomplete datasets. This sensitivity also varies based on both the completeness level of datasets and gap sizes. In general, as we increase completeness level, the accuracy increases, and vice versa. The same thing might apply to gap lengths, as we increase them, accuracy tends to decrease; however, not if the datasets are well reconstructed. For completed datasets, E-KNN performance is often somehow stable under different missing percentages and gap lengths; however, sometimes these two parameters also impact E-KNN's performance in a similar manner as happens with incomplete datasets. When the performance of E-KNN is stable with regard to different gaps and completeness ratios, it means that the imputation techniques are efficient and well reconstructed the datasets despite the different factors considered.

6. Conclusions

We investigated in this paper the impact of data loss on the performance of the K-nearest neighbors model applied to the context of intelligent transportation. The model delivers a multi-step flow forecast for urban roads located in downtown Bremen. In order to examine the efficiency of our E-KNN model under data loss circumstances, we artificially created incomplete datasets with different completeness levels and gap lengths. Afterward, we designed three different methods for the sake of reconstructing these datasets. The performance of the E-KNN model is then tested with the original, incomplete, and imputed datasets. The experimental results showed that E-KNN is able to reach 17% MAPE during significant daily traffic (06:00–22:00) for 3 weeks of the test set. Moreover, considering all day hours, the model is able to make 6-step forecasts with an average error of one car per 90 s. The performance of the model under imputed datasets varies in function of completeness level and gap length, but in general the model performs much better under filled-in datasets than under incomplete ones.

Author Contributions: Conceptualization, A.M. and D.K.; Funding acquisition, C.B.; Methodology, A.M. and D.K.; Project administration, C.B.; Supervision, C.B.; Writing—original draft, A.M.; Writing—review & editing, A.M. and D.K. All authors have read and agreed to the published version of the manuscript.

Funding: DiSCO2 project is funded by the European Regional Development Fund (ERDF).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kumar, S.V.; Vanajakshi, L. Short-term traffic flow prediction using seasonal ARIMA model with limited input data. *Eur. Transp. Res. Rev.* **2015**, *7*, 21 [CrossRef]
2. Duan, P.; Mao, G.; Zhang, C.; Wang, S. STARIMA-based traffic prediction with time-varying lags. In Proceedings of the 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), Rio de Janeiro, Brazil, 1–4 November 2016; pp. 1610–1615.
3. Van Der Voort, M.; Dougherty, M.; Watson, S. Combining Kohonen maps with ARIMA time series models to forecast traffic flow. *Transp. Res. Part C Emerg. Technol.* **1996**, *4*, 307–318. [CrossRef]
4. Lu, S.; Zhang, Q.; Chen, G.; Seng, D. A combined method for short-term traffic flow prediction based on recurrent neural network. *Alex. Eng. J.* **2021**, *60*, 87–94. [CrossRef]
5. Sadeghi-Niaraki, A.; Mirshafiei, P.; Shakeri, M.; Choi, S.M. Short-Term Traffic Flow Prediction Using the Modified Elman Recurrent Neural Network Optimized Through a Genetic Algorithm. *IEEE Access* **2020**, *8*, 217526–217540. [CrossRef]
6. Bai, L.; Yao, L.; Li, C.; Wang, X.; Wang, C. Adaptive Graph Convolutional Recurrent Network for Traffic Forecasting. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 17804–17815.
7. Lv, Y.; Duan, Y.; Kang, W.; Li, Z.; Wang, F. Traffic Flow Prediction with Big Data: A Deep Learning Approach. *IEEE Trans. Intell. Transp. Syst.* **2015**, *16*, 865–873. [CrossRef]
8. Klosa, D.; Mallek, A.; Büskens, C. Short-Term Traffic Flow Forecast Using Regression Analysis and Graph Convolutional Neural Networks. In Proceedings of the 2021 IEEE 23rd International Conference on High Performance Computing & Communications; 7th International Conference on Data Science & Systems; 19th International Conference on Smart City; 7th International Conference on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys), Hainan, China, 20–22 December 2021; pp. 1413–1418.
9. Castro-Neto, M.; Jeong, Y.S.; Jeong, M.K.; Han, L.D. Online-SVR for short-term traffic flow prediction under typical and atypical traffic conditions. *Expert Syst. Appl.* **2009**, *36*, 6164–6173. [CrossRef]
10. Liu, L. A Short-Term Traffic Flow Prediction Method Based on SVR. In Proceedings of the 2021 2nd International Conference on Urban Engineering and Management Science (ICUEMS), Sanya, China, 29–31 January 2021; pp. 1–4.
11. Zheng, Z.; Su, D. Short-term traffic volume forecasting: A k-nearest neighbor approach enhanced by constrained linearly sewing principle component algorithm. *Transp. Res. Part C Emerg. Technol.* **2014**, *43*, 143–157. [CrossRef]
12. Cai, P.; Wang, Y.; Lu, G.; Chen, P.; Ding, C.; Sun, J. A spatiotemporal correlative k-nearest neighbor model for short-term traffic multistep forecasting. *Transp. Res. Part C Emerg. Technol.* **2016**, *62*, 21–34. [CrossRef]
13. Cheng, S.; Lu, F.; Peng, P.; Wu, S. Short-term traffic forecasting: an adaptive ST-KNN model that considers spatial heterogeneity. *Comput. Environ. Urban Syst.* **2018**, *71*, 186–198. [CrossRef]
14. Nihan, N.L. Aid to determining freeway metering rates and detecting loop errors. *J. Transp. Eng.* **1997**, *123*, 454–458. [CrossRef]
15. Zhong, M.; Lingras, P.; Sharma, S. Estimation of missing traffic counts using factor, genetic, neural, and regression techniques. *Transp. Res. Part C Emerg. Technol.* **2004**, *12*, 139–166. [CrossRef]
16. Liu, Z.; Sharma, S.; Datla, S. Imputation of missing traffic data during holiday periods. *Transp. Plan. Technol.* **2008**, *31*, 525–544. [CrossRef]
17. Tian, Y.; Zhang, K.; Li, J.; Lin, X.; Yang, B. LSTM-based traffic flow prediction with missing data. *Neurocomputing* **2018**, *318*, 297–305. [CrossRef]
18. Duan, Y.; Lv, Y.; Liu, Y.L.; Wang, F.Y. An efficient realization of deep learning for traffic data imputation. *Transp. Res. Part C Emerg. Technol.* **2016**, *72*, 168–181. [CrossRef]
19. Pamuła, T. Impact of data loss for prediction of traffic flow on an urban road using neural networks. *IEEE Trans. Intell. Transp. Syst.* **2018**, *20*, 1000–1009. [CrossRef]
20. Qu, L.; Li, L.; Zhang, Y.; Hu, J. PPCA-based missing data imputation for traffic flow volume: A systematical approach. *IEEE Trans. Intell. Transp. Syst.* **2009**, *10*, 512–522.
21. Li, L.; Li, Y.; Li, Z. Efficient missing data imputing for traffic flow by considering temporal and spatial dependence. *Transp. Res. Part C Emerg. Technol.* **2013**, *34*, 108–120. [CrossRef]
22. Bishop, C. Bayesian pca. *Adv. Neural Inf. Process. Syst.* **1998**, *11*.
23. Minka, T. Automatic choice of dimensionality for PCA. *Adv. Neural Inf. Process. Syst.* **2000**, *13*.
24. Muralidharan, A.; Horowitz, R. Imputation of ramp flow data for freeway traffic simulation. *Transp. Res. Rec.* **2009**, *2099*, 58–64. [CrossRef]
25. Van Lint, J.; Hoogendoorn, S.; van Zuylen, H.J. Accurate freeway travel time prediction with state-space neural networks under missing data. *TRansportation Res. Part C Emerg. Technol.* **2005**, *13*, 347–369. [CrossRef]

26. Chen, H.; Grant-Muller, S.; Mussone, L.; Montgomery, F. A study of hybrid neural network approaches and the effects of missing data on traffic forecasting. *Neural Comput. Appl.* **2001**, *10*, 277–286. [CrossRef]
27. Tang, J.; Zhang, G.; Wang, Y.; Wang, H.; Liu, F. A hybrid approach to integrate fuzzy C-means based imputation method with genetic algorithm for missing traffic volume data estimation. *Transp. Res. Part C Emerg. Technol.* **2015**, *51*, 29–40. [CrossRef]
28. Tan, H.; Feng, G.; Feng, J.; Wang, W.; Zhang, Y.J.; Li, F. A tensor-based method for missing traffic data completion. *Transp. Res. Part C Emerg. Technol.* **2013**, *28*, 15–27. [CrossRef]
29. Wang, F.Y. Parallel control and management for intelligent transportation systems: Concepts, architectures, and applications. *IEEE Trans. Intell. Transp. Syst.* **2010**, *11*, 630–638. [CrossRef]
30. Smith, B.L.; Demetsky, M.J. Traffic flow forecasting: comparison of modeling approaches. *J. Transp. Eng.* **1997**, *123*, 261–266. [CrossRef]
31. Smith, B.L.; Williams, B.M.; Oswald, R.K. Comparison of parametric and nonparametric models for traffic flow forecasting. *Transp. Res. Part C Emerg. Technol.* **2002**, *10*, 303–321. [CrossRef]
32. Davis, G.A.; Nihan, N.L. Nonparametric regression and short-term freeway traffic forecasting. *J. Transp. Eng.* **1991**, *117*, 178–188. [CrossRef]
33. Habtemichael, F.G.; Cetin, M.; Anuar, K.A. Methodology for quantifying incident-induced delays on freeways by grouping similar traffic patterns. In Proceedings of the Transportation Research Board 94th Annual Meeting, Washington, DC, USA, 11–15 January 2015; pp. 15–4824.

Short-Term Traffic Flow Forecast Using Regression Analysis and Graph Convolutional Neural Networks

Daniel Klosa
Center for Industrial Mathematics
University of Bremen
Bremen, Germany
dklosa@uni-bremen.de

Amin Mallek
Center for Industrial Mathematics
University of Bremen
Bremen, Germany
amallek@uni-bremen.de

Christof Büskens
Center for Industrial Mathematics
University of Bremen
Bremen, Germany
bueskens@math.uni-bremen.de

***Abstract*—Short-term forecast of different traffic attributes is one of the fundamental tools used in transportation planning. Precisely, accurate predictions of traffic flow are often the basis of an efficient traffic management. In the paper at hands we shed a light on short-term traffic flow forecast in urban arterial roads in Bremen (Germany). This case-study uses real-data collected from 7 loop detectors installed in downtown. To deal with this, we propose two different models, namely, Linear Regression and Graph Convolutional Neural Networks. The models are separately applied to 11 weeks of data, three of these are dedicated to test the performance of both models. Experimental results show that the models are closely competitive and they reach, in average over the whole test-set, around 19% mean absolute percentage error during morning and evening peak times.**

***Index Terms*—Traffic Flow; Regression Analysis; Graph Convolutional Neural Networks; Short-term Forecast; Deep Learning; Machine Learning.**

I. Introduction

Intelligent transportation systems (ITS) play a significant role in organizing, controlling and planning the traffic flow inside or outside of cities. Basically, smart cities heavily rely on those systems to optimize the circulation process and render city districts easily and quickly accessible from everywhere. One of the main elements ITS is concerned with is traffic flow volume in different city roads and further highways surrounding the city. Various tools are usually used to measure and monitor the traffic flow including inductive loop detectors, video image processing, microwave and laser radars and other techniques related to Internet of Things (IoT). The data collected by the aforementioned tools (which is usually Big Data), is often used to build different types of models for analysis and prediction purposes. Many researchers investigated broad topics of Intelligent Transportation with a major focus on predicting the key factors that impact the traffic, for instance, traffic flow volume, traffic speed, traffic state, etc. In the present paper we draw attention to short-term traffic flow volume forecasting. Accurate prediction of this key element is a stepping stone for other kinds of tasks such as travel time forecast and traffic state prediction.

Funded by the European Regional Development Fund (ERDF).

European Union
Investing in Bremen's Future
European Regional
Development Fund

In our study, we use two different methods to deal with short-term forecasting of traffic flow volume in Bremen city (Germany). This is part of a project we conduct to model and forecast traffic flow in the city, wherein the chief goal is to reduce CO_2 emissions in Bremen by controlling traffic lights. In order to capture the linear relations in the working data, we designed a Regression Analysis model based on different time frames aiming to learn the linear dependencies between data points. To cope with the non-linear characteristics in the data, a Graph Convolutional Neural Network is used. For the time being, both models were separately applied to data collected from 7 inductive loop detectors installed around urban arterial roads in downtown. The purpose of this paper is to compare the proposed models, evaluate the prediction results obtained so far and give insights on some future work. The rest of this article is structured as follows. We review some related work in Section II. In Section III, we detail the models devised in the paper. Afterwards, the used data is described in Section IV. Results and discussions are reported in Section V. The paper is concluded with some future directions and perspectives in Section VI.

II. Literature review

The problem we consider has gained a lot of interest among researchers and has been investigated in many papers with different approaches. One of the common approaches often used when one deals with time series is ARIMA and its variants. This model has been solely applied as in [9] and also hybridized with some other approaches such as in [15]. Moreover, other techniques such as Support Vector Regression (SVR) [11], [3], K-Nearest Neighbors (KNN) [2], [17], [19] and several hybridized methods [10], [18] were also explored. However, most recently researchers focus more on Deep Learning methods to improve forecasting precision.

Some of the early Deep Learning models such as stacked auto encodes (SAE) [12], long-short term memory (LSTM) and gated recurrent units (GRU) [5] used only time series data in the prediction process. Nevertheless, recent models employ more advanced techniques as in [7], where Traffic-Wave is used. The authors stack 1d dilated convolutions to increase the receptive field of the convolutions exponentially, which allows their model to extract short-term patterns in lower layers and long-term patterns in higher layers, outperforming the previous approaches. In Graph Wave-Net, as in [14], the Traffic-Wave idea is extended by implementing graph convolutions to capture spatial dependencies. Most recent Deep Learning techniques for traffic forecasting incorporate graph convolutions [1], [4], [6]. For an overview, the reader is referred to a survey that has been carried out in [8].

III. Methodology

This section comprises a detailed description of the forecasting models designed in the current article.

A. Regression Analysis

Linear Regression (precisely Polynomial Regression in our case), is a modeling approach that captures a certain relationship between variables, usually known as independent variables or predictors (input) and dependent variables (output). In cases when we deal with only one input variable the model is called Simple Linear Regression, however, when the number of independent variables escalates, we refer to the model as Multiple Linear Regression. Another variant of linear regression models, different than the latter, named Multivariate Linear Regression, also exists in the literature and is often involved when multiple dependent variables (output) are to be predicted. The relationships between the different variables are modeled using linear predictor functions whose unknown model parameters are generally estimated from the data.

Linear Regression models have been extensively used in various areas for multiple goals. Here we are interested in their applications in the field of Machine Learning where the main aim is to build a predictive model from an observed data-set, then use the learnt model to predict or forecast future (unseen) values. The general equation of a polynomial regression of degree n can be written (in two forms) as follows:

$$P(x) = \sum_{k=0}^{n} \alpha_k x^k + \epsilon = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \ldots + \alpha_n x^n + \epsilon$$

$P(x)$ and x are respectively the dependent variable and the independent variable. α is the vector of the corresponding coefficients to be learnt from the data, and ϵ is a random error term with mean zero; added for bias.
Our model is devised in 2-phases, the first one is to establish a Regression Analysis, then subdue the model parameters, in a second part, to a correction or a validation process in which the values are slightly modified. The regression model considers each week-day separately, for each of which a polynomial of degree 10 is learnt, this yields 7 functions. Moreover, each of the week-days is split up into 24 hours, to each hour a polynomial of degree 5 is associated. Thus, in total we have 175 functions (168+7). The Regression Analysis takes into account the mean of the values at a given instant over the training set, then excludes the values that are above $1.5 \cdot mean$ since polynomial regression is very sensitive to outliers. The shape of the daily regression is smoothly captured with degree 10 with no over-fitting. Additionally, degree 5 is chosen for the hourly regression as an attempt to grasp the shape of the fluctuations and try to match with it. The predicted flow values at a given instant are calculated as follows:

$$F(t) = \frac{4 \cdot F_{week-day}(t) + F_{hour}(t)}{5}$$

where $F(t)$ is the predicted traffic flow volume at instant t, $F_{week-day}(t)$ and $F_{hour}(t)$ are respectively the regression polynomials corresponding to the week-day and the hour of instant t. The correction phase gives high priority to the parameters learnt by the regression model, however it tries to make use of one week values and combine them with the regression values as follows:

$$Pred(t) = \frac{4 \cdot V_{reg}(t) + V_{week}(t)}{5}$$

Where $V_{week}(t)$ are the values at each instant t of the last week-days of the training set and $V_{reg}(t)$ are the output of the regression model at the same instant t. Experimentally, this latter showed a slight improvement in the prediction accuracy.

The built-up model has the ability to do short and long term forecasts very quickly, alongside that, it can be trained with a small amount of data. In contrast, the main drawbacks of the model are that it cannot accurately predict the sharp fluctuations without further data feeding and also is unable to predict any strange pattern that was not captured during the training phase.

B. Graph Convolutional Neural Network

The Graph Convolutional Neural Network (GCNN) used herein is based on the Global Spatial-Temporal Graph Convolutional Network (GSTGCN) by Liang et al. [6]. The architecture used in the present work is illustrated in Figure 1. The three temporal modules are each made up of $N_R = 3$ residual blocks incorporating dilated 1d convolutions (DCC) with dilation factors $d = 1, 2, 4$, in the same way as in [16], for learning short-term and long-term relationships in the temporal space. Dilated convolution blocks were also used in Traffic-Wave [7] and Graph Wave-Net [14]. They are followed by a weight normalization operation (WN) [13] to combat overfitting and a ReLU activation.

We use three input segments for each prediction; Let $x_{t_0} \in \mathbb{R}^{N \times D}$ be the traffic state at the current timestamp t_0, where $N = 7$ is the amount of sensors, $D = 1$ is

the amount of features (in this case flow) and $T_p \in \mathbb{N}$ is the amount of timestamps to be predicted. The recent-time segment $X_{recent} = (x_{t_0-T_h+1}, x_{t_0-T_h+2}, \ldots, x_{t_0}) \in \mathbb{R}^{N\times D\times T_h}, T_h \in \mathbb{N}$ includes the most recent timestamps and it is used to adapt to the current traffic state. The daily-periodic segment $X_{daily} \in \mathbb{R}^{N\times D\times T_d}$, $T_d \in \mathbb{N}$ incorporates the same segment to be predicted but on the previous $\frac{T_d}{T_p} \in \mathbb{N}$ days. This segment is chosen because traffic can be similar on consecutive days. The weekly-periodic segment $X_{weekly} \in \mathbb{R}^{N\times D\times T_w}, T_w \in \mathbb{N}$ also contains the same segment to be predicted but from the previous $\frac{T_d}{T_p} \in \mathbb{N}$ weeks on the same weekday. It makes use of repeating patterns in the historical data since the flow on the same weekday is generally similar. The three temporal modules are used for learning recent, daily-periodic and weekly-periodic features $Y_{recent}, Y_{daily}, Y_{weekly} \in \mathbb{R}^{N\times F\times T_p}$, where $N = 7$ is the amount of sensors, $F \in \mathbb{N}$ is the amount of features and $T_p \in \mathbb{N}$ is the amount of timestamps to be predicted. The spatial modules consist of a graph convolution incorporating the distance between sensors and a global correlated spatial mechanism based on the connectivity within the traffic network. The adjacency matrix deployed in the graph convolution is based on driving distances in between sensors. Accordingly, Dijkstra algorithm is applied to compute the driving distance $dist(i, j)$ from sensor i to sensor j. For the global correlated spatial mechanism, the connectivity $conn(i, j)$ of two sensors i and j is set to $conn(i, j) = \alpha = 2$ if there are at most two edges in between sensors i and j, otherwise $conn(i, j) = 1$. The resulting features of the three time-frames are fused and a fully connected neural network is applied. While Liang et al. in [6] use GSTGCN to predict traffic speed, the adapted architecture demonstrates through experimental testing that it is also capable to predict traffic flow.

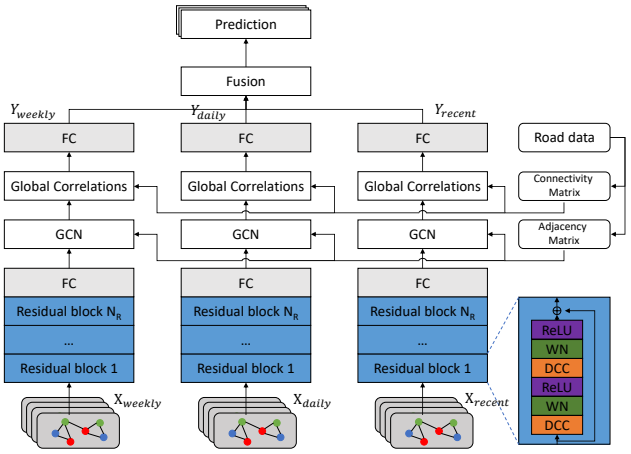


Fig. 1: The architecture of our Graph Convolutional Neural Network.

IV. Data description

The present paper is part of a project currently conducted at the Center for Industrial Mathematics (ZeTeM) of the University of Bremen, to model and forecast traffic flow in the city of Bremen (Germany). The project is launched in the context of the increased efforts to combat climate change. As stated above, the main goal of this project is to forecast a short-term traffic state to allow decision-makers to take actions in order to reduce CO_2 emissions due to traffic. The Traffic Management Center (VMZ) of Bremen is an associated partner in the project and it is the main provider of the data we use. Figure 2 presents Bremen city map and displays in red bullets the location of the detectors installed all over the city to record traffic data.

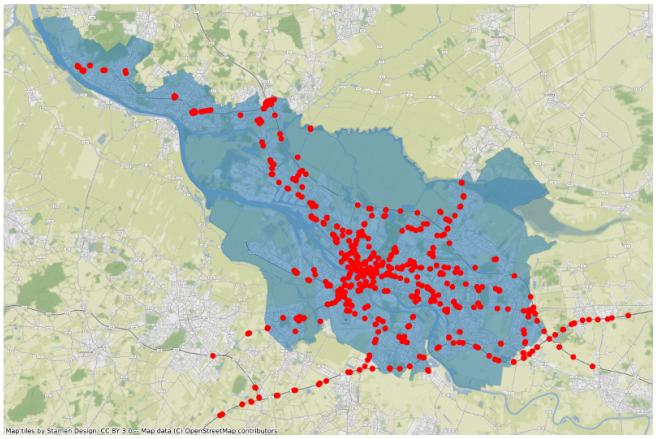


Fig. 2: Loop detectors installed in the city of Bremen.

In the study reported in this paper, we focus on probably the most congested area in the city: downtown. We selected a junction in front of the main train station (Bremen Hauptbahnhof), tram station, bus station, etc. The chosen junction is itself a

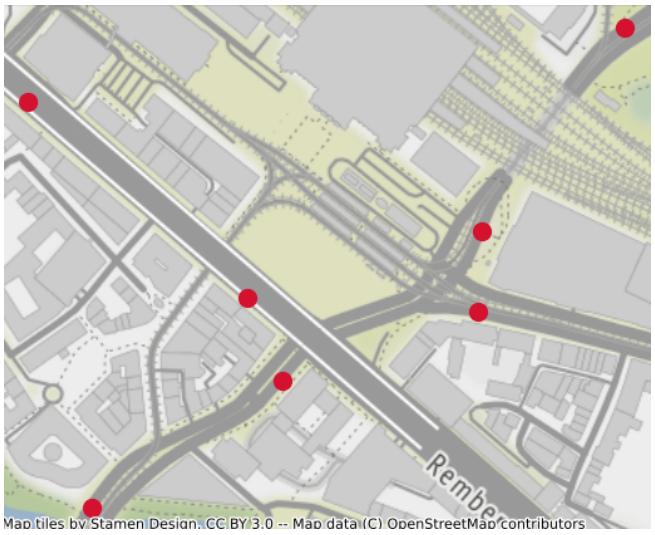


Fig. 3: The chosen junction (Red bullets correspond to the detectors location).

big challenge because of the large amount of traffic lights and signs present in this area, about which, unfortunately, we don't

possess any data. In that location 7 inductive loop detectors (form MS217 to MS223) are installed in the surroundings to collect traffic data as shown in Figure 3.

Measurements take place every 90 seconds, however, in our study we use 10-minutes accumulation. The sample data we consider in the present work includes the time frame from 9 April 2018 to 24 June 2018. The chosen dates correspond to the period with the least missing data (less than 2% of the data is missing). The used data covers a period of 11 weeks split up into 8 weeks for training the models and 3 weeks for testing. Precisely, the models are trained with data from 9 April 2018 to 3 June 2018 and tested for the period going from 4 June 2018 to 24 June 2018. Figures 4 and 5 exhibit random samples of data per day and per week.

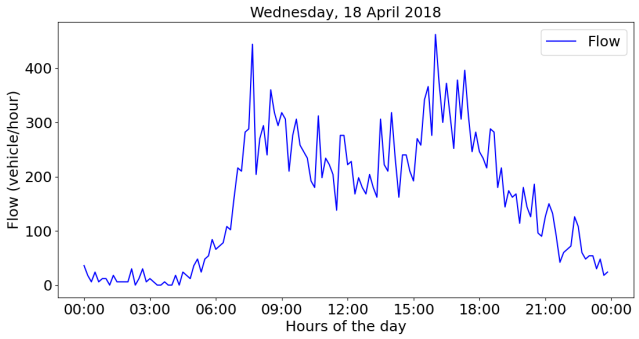


Fig. 4: One-day traffic flow data (detector MS223).

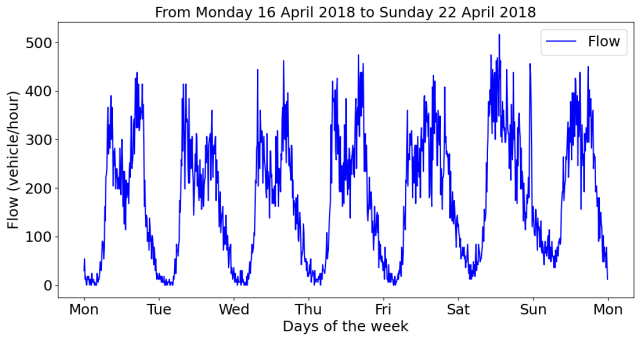


Fig. 5: One-week traffic flow data (detector MS223).

Since the data is collected from detectors placed in signalized urban arterial roads, the fluctuations in the data are very sharp and can jump between extreme values in short periods of time. In our project, we focus on different attributes related to traffic and transportation, however, in this paper we discuss only traffic flow forecast. Note that traffic flow volume is given by the number of vehicles traversing a detector per hour.

V. Results and Discussions

In this section, we report and discuss the output of the implemented models. Before doing so, in order to evaluate the accuracy of the predictions made by both models, two metrics are deployed: Mean Absolute Percentage Error (MAPE) and Mean Absolute Error (MAE). These are given by the following formulas:

$$MAPE = \frac{100}{n} \cdot \sum_{i=1}^{n} \left| \frac{m(t)_i - p(t)_i}{m(t)_i} \right|$$

$$MAE = \frac{1}{n} \cdot \sum_{i=1}^{n} |m(t)_i - p(t)_i|$$

such that $m(t)$ is the real value of traffic flow measured at instant t and $p(t)$ is the value predicted by the model. n is the number of predictions.

TABLE I: Prediction results over the test-set considering morning peak hours only (06:00-09:00).

	MAPE(%)		MAE	
Detector ID	Reg. Ana.	GCNN	Reg. Ana.	GCNN
MS217	15.50	16.11	78.93	65.65
MS218	25.27	34.15	25.39	28.14
MS219	16.50	13.74	75.86	56.65
MS220	20.09	20.02	77.47	73.36
MS221	22.28	30.34	26.11	32.07
MS222	22.21	27.08	39.50	40.20
MS223	23.46	20.93	53.56	47.12
Average	20.75	23.19	53.83	49.02

TABLE II: Prediction results over the test-set considering evening peak hours only (16:00-19:00).

	MAPE(%)		MAE	
Detector ID	Reg. Ana.	GCNN	Reg. Ana.	GCNN
MS217	11.77	11.77	96.81	96.02
MS218	20.85	19.90	58.78	48.26
MS219	11.04	11.22	84.80	86.62
MS220	17.19	15.38	79.51	71.50
MS221	19.81	18.98	57.56	55.46
MS222	20.01	19.48	61.91	59.05
MS223	18.64	17.31	52.98	48.67
Average	17.04	16.29	70.33	66.51

TABLE III: Prediction results over the test-set considering all-day hours.

	MAPE(%)		MAE	
Detector ID	Reg. Ana.	GCNN	Reg. Ana.	GCNN
MS217	15.81	17.03	63.05	61.95
MS218	28.30	32.13	28.81	29.02
MS219	18.01	19.62	60.22	58.12
MS220	24.19	25.19	61.39	58.95
MS221	29.49	29.69	38.98	36.67
MS222	29.67	30.42	39.91	38.15
MS223	27.91	27.80	37.92	34.99
Average	24.76	25.98	47.18	45.40

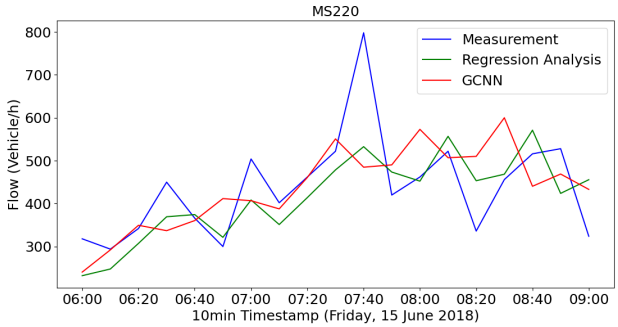


Fig. 6: Flow prediction in morning peak on a randomly chosen working-day (detector MS220)

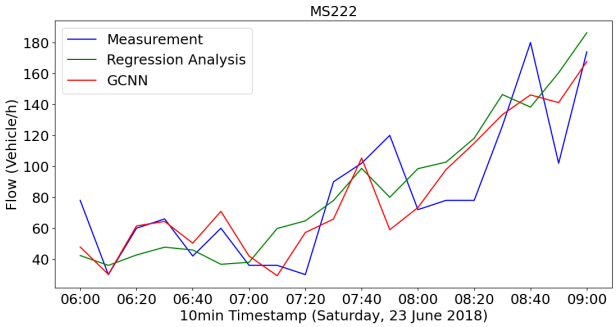


Fig. 7: Flow prediction in morning peak on a randomly chosen weekend-day (detector MS222).

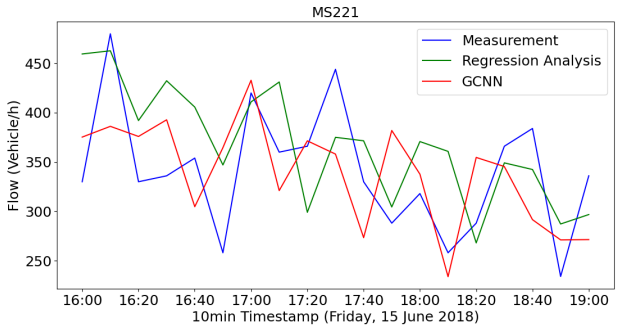


Fig. 8: Flow prediction in evening peak on a randomly chosen working-day (detector MS221).

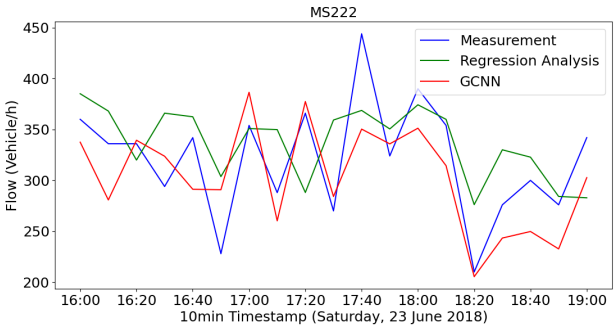


Fig. 9: Flow prediction in evening peak on a randomly chosen weekend-day (detector MS222).

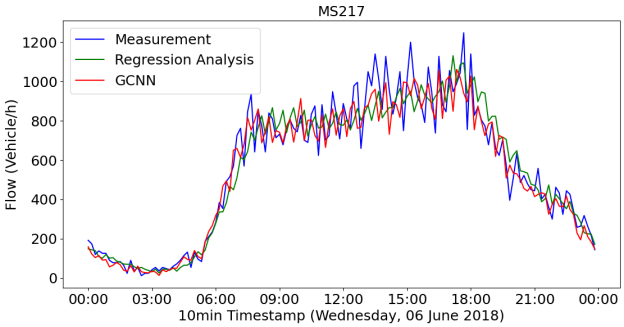


Fig. 10: Flow prediction on a randomly chosen working-day (detector MS217)

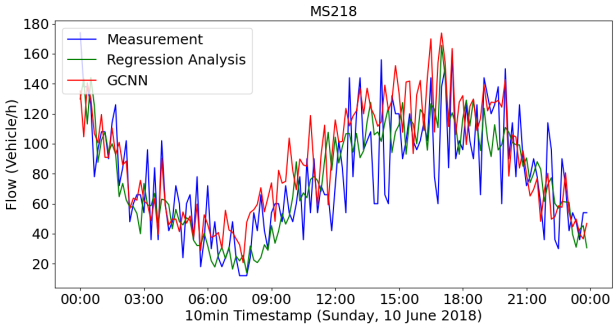


Fig. 11: Flow prediction on a randomly chosen weekend-day (detector MS218)

The discussion of the outcome of the Regression Analysis and the Graph Convolutional Neural Network (GCNN) models is carried out according to all-day hours and traffic peak hours in the morning and in the evening. We report the results of 3 weeks of testing from 4 June 2018 to 24 June 2018. The focus is on morning and evening peaks where a significant amount of vehicles is flowing. The morning peak is set to be between 06:00 and 09:00. In the evening, 3 hours of traffic peak is chosen between 16:00 and 19:00. The results are exhibited per detector as shown in Tables I, II and III.

Tables I and II recapitulate the results of the experimental tests per detector for the morning and evening peaks respectively. The Regression Analysis model is applied to each detector's data separately, thus, the learnt functions are in total 1225. However, GCNN takes into account all detectors data simultaneously. For GCNN we set a one-hour forecast horizon, whereas the Linear Regression model performs one-day prediction at a time. We can clearly see that the prediction accuracy differs from one detector to another. This mainly pertains to the flow volume where the detector is placed,

because some of them records from multiple-lane roads (roads do not have the same number of lanes). When the flow volume is high, the MAE tends to take higher values, and vice versa. This also is the reason why we have higher MAE during peak hours than all-day hours. Athwart, the MAPE, measuring a relative error, decreases during peak times. In average, both models produce $24\% - 25\%$ MAPE for all-day hours, which decreases to around 19% during morning and evening peaks. As mentioned above, since we are dealing with a signalized junction situated in downtown, the reached accuracy is satisfactory, especially during the most congested times of the day. It is noteworthy that our data differs from the one used in recent works, wherein data-sets are mostly from the Caltrans Performance Measure System (PeMS). The just-mentioned system records highway traffic data in California, which lacks the strong uncertainty about a vehicle's behavior (taken direction) at a given junction. Furthermore, apart from the noise created by traffic light signals, this kind of behaviour constantly changes and is very hard to predict.

For visualization purposes, Figures 10, 11, 6, 7, 8 and 9 illustrate the performance of both models on randomly chosen detectors on different days (weekend and working-days). Although the exhibited samples show that the models can accurately track the measured flow trend, they also display the weakness of the models, which is the inability of predicting occasional peaks (such as in Figure 6 at 07:40). Another data-related weakness, is the sharp fluctuations that appear from time to time (refer to Figure 11). As mentioned above, unfortunately, we don't have any data about these unusual traffic patterns.

VI. Conclusion and future directions

This article dealt with forecasting traffic flow volume at a single signalized traffic junction in Bremen (Germany). We focused on a very congested location, in which 7 loop detectors are installed. In order to forecast traffic flow in this region, we proposed two different models: Linear Regression and Graph Convolutional Neural Network based models. On a 3 weeks test-set, we measured the accuracy of both models deploying MAE and MAPE as performance indicators. The experimental results showed that the models are closely competitive and produce satisfactory forecasts with 19% error in average for the peak times during the day and with averagely 25% for all-day hours. To improve upon this, we plan to benefit from both models strength, and possibly, hybridize between them. Additionally, collecting traffic light data could help in predicting the fluctuating traffic patterns.

References

[1] Lei Bai, Lina Yao, Can Li, Xianzhi Wang, and Can Wang. Adaptive graph convolutional recurrent network for traffic forecasting. *Advances in Neural Information Processing Systems*, 33, 2020.

[2] Pinlong Cai, Yunpeng Wang, Guangquan Lu, Peng Chen, Chuan Ding, and Jianping Sun. A spatiotemporal correlative k-nearest neighbor model for short-term traffic multistep forecasting. *Transportation Research Part C: Emerging Technologies*, 62:21–34, 2016.

[3] Manoel Castro-Neto, Young-Seon Jeong, Myong-Kee Jeong, and Lee D Han. Online-svr for short-term traffic flow prediction under typical and atypical traffic conditions. *Expert systems with applications*, 36(3):6164–6173, 2009.

[4] Jun Fu, Wei Zhou, and Zhibo Chen. Bayesian graph convolutional network for traffic prediction, 2021.

[5] Rui Fu, Zuo Zhang, and Li Li. Using lstm and gru neural network methods for traffic flow prediction. In *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, pages 324–328, 2016.

[6] Liang Ge, Siyu Li, Yaqian Wang, Feng Chang, and Wu Kunyan. Global spatial-temporal graph convolutional network for urban traffic speed prediction. *Applied Sciences*, 10:1509, 02 2020.

[7] Donato Impedovo, Vincenzo Dentamaro, Giuseppe Pirlo, and Lucia Sarcinella. Trafficwave: Generative deep learning architecture for vehicular traffic flow prediction. *Applied Sciences*, 9(24):5504, 2019.

[8] Weiwei Jiang and Jiayun Luo. Graph neural network for traffic forecasting: A survey. *arXiv preprint arXiv:2101.11174*, 2021.

[9] S Vasantha Kumar and Lelitha Vanajakshi. Short-term traffic flow prediction using seasonal arima model with limited input data. *European Transport Research Review*, 7(3):1–9, 2015.

[10] Wan Li, Jingxing Wang, Rong Fan, Yiran Zhang, Qiangqiang Guo, Choudhury Siddique, and Xuegang Jeff Ban. Short-term traffic state prediction from latent structures: Accuracy vs. efficiency. *Transportation Research Part C: Emerging Technologies*, 111:72–90, 2020.

[11] Longshun Liu. A short-term traffic flow prediction method based on svr. In *2021 2nd International Conference on Urban Engineering and Management Science (ICUEMS)*, pages 1–4. IEEE, 2021.

[12] Yisheng Lv, Y. Duan, Wenwen Kang, Zhengxi Li, and F. Wang. Traffic flow prediction with big data: A deep learning approach. *IEEE Transactions on Intelligent Transportation Systems*, 16:865–873, 2015.

[13] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[14] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. Graph wavenet for deep spatial-temporal graph modeling, 2019.

[15] Hanyu Yang, Xutao Li, Wenhao Qiang, Yuhan Zhao, Wei Zhang, and Chang Tang. A network traffic forecasting method based on sa optimized arima–bp neural network. *Computer Networks*, 193:108102, 2021.

[16] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions, 2015.

[17] Lun Zhang, Qiuchen Liu, Wenchen Yang, Nai Wei, and Decun Dong. An improved k-nearest neighbor model for short-term traffic flow prediction. *Procedia-Social and Behavioral Sciences*, 96:653–662, 2013.

[18] Yanru Zhang, Yunlong Zhang, and Ali Haghani. A hybrid short-term traffic flow forecasting method based on spectral analysis and statistical volatility model. *Transportation Research Part C: Emerging Technologies*, 43:65–78, 2014.

[19] Zuduo Zheng and Dongcai Su. Short-term traffic volume forecasting: A k-nearest neighbor approach enhanced by constrained linearly sewing principle component algorithm. *Transportation Research Part C: Emerging Technologies*, 43:143–157, 2014.

Evolutionary Neural Architecture Search for Traffic Forecasting

Daniel Klosa
Center for Industrial Mathematics
University of Bremen
Bremen, Germany
dklosa@uni-bremen.de

Christof Büskens
Center for Industrial Mathematics
University of Bremen
Bremen, Germany
bueskens@math.uni-bremen.de

***Abstract*—Traffic forecasting is a challenging task due to complex spatial and temporal dependencies across sensor locations and time. Interest in solving this task has increased, but current research focuses on manually constructing neural network architectures without the aid of neural architecture search (NAS). In our work, we explore evolutionary neural architecture search (ENAS) by deploying a genetic algorithm (GA) to find optimal neural network architectures for predicting traffic conditions. The search space for the GA consists of arbitrary combinations of dilated convolutions and graph convolutions for modelling temporal and spatial dependencies respectively, limited in complexity only by technical constraints. Experimental results show that model architectures obtained via GA are able to match the current state-of-the-art on traffic prediction benchmarks.**

***Index Terms*—evolutionary neural architecture search, genetic algorithm, neural architecture search, traffic forecasting, deep learning**

I. Introduction

Traffic forecasting is the task of predicting future traffic conditions, such as flow and speed, by analyzing historical traffic patterns. These forecasts are useful for detecting congestion and long travel time risk, can help authorities in planning and controlling traffic as well as help citizens to make routing decisions. Furthermore, especially in urban and metropolitan areas, enabling intelligent traffic systems (ITS) to adjust to future events can lead to more homogeneous traffic flow, and hence, fewer CO_2 emissions, reducing environmental impact. Since traffic data has become more available in recent years, there has been increasing research towards developing machine learning algorithms for traffic forecasting.

Traffic forecasting is a challenging task, since the future traffic conditions at one measurement site do not only depend on the recent conditions (temporal dimension), but also on the ones from upstream measurement sites (spatial dimension). Furthermore, datasets can contain hundreds of measurement sites and the underlying road network can be complex, making it difficult to model dependencies between them.

Traditional methods such as linear regression [18], auto-regressive moving average [14] and vector auto-regression [24] fail to capture the complex spatio-temporal dependencies of large traffic datasets. Hence, recent research has shifted towards deep learning related models such as long-short term memory and gated recurrent unit [4] for capturing temporal dependencies and graph convolutional neural networks (GCN) to learn the spatial dependencies within the data [6], [8], [21]. One drawback of GCN models is that they require knowledge of spatial connections within the graph structured road network, often in the form of an adjacency matrix. Graph WaveNet [21] and AGCRN [2] solve this issue by learning the spatial dependencies directly from the data. However, the neural architectures used are still handcrafted by experts. When deployed in real world applications, these approaches additionally need to be tailored to the scenario at hand, requiring a considerable amount of time and effort.

Alleviating this tedious process of neural architecture design, neural architecture search (NAS) methods have become a popular means for discovering tailored neural architectures for various tasks. Early NAS frameworks focus on computer vision and language modelling [11], [16], however, they can also be applied to graph data [5], [23] and spatio-temporal data [15]. There are three common components in NAS. The first one being the search space, i.e. the general structure of the discovered network architectures, which is defined by the operations and their connections within the network. Secondly, there are different search strategies, the main ones are reinforcement learning (RL), gradient-based search and evolutionary NAS (ENAS). RL based algorithms often require a tremendous amount of computation time, even on smaller datasets such as CIFAR-10 [25]. Gradient-based NAS as in [11] are more efficient than RL based methods. However, they can get stuck in local minima and require constructing a supernet in advance, which constrains the search space and necessitates multiple searches for different hyperparameter settings, e.g. number of cells, number of vertices in cells and hidden units. Although ENAS can also suffer from long computation times, they can explore the search space more thoroughly without a given supernet in advance. Lastly, the performance estimation strategy defines how discovered architectures are evaluated. Often, this is done by training them for a certain amount of epochs. However, since this is costly, strategies like weight inheritance and network morphisms

Funded by the European Regional Development Fund (ERDF).

European Union
Investing in Bremen's Future
European Regional
Development Fund

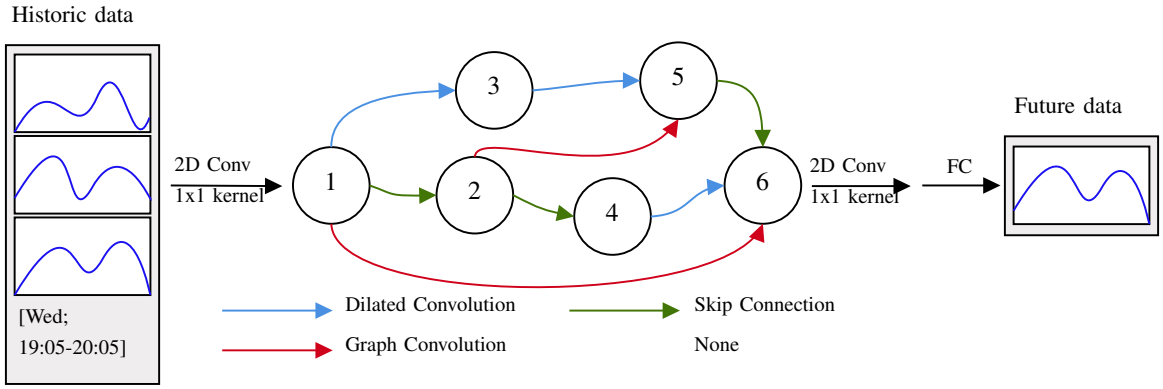


Fig. 1: Architecture search space of our framework.

eliminate the need for training from scratch, greatly reducing training time [3]. It is also possible to evaluate discovered architectures without training at all [12].

The research on NAS for traffic forecasting is limited. Early work [20] investigates the implementation of genetic algorithm (GA) for optimizing gradient descent hyperparameters and hidden layers in multi layer perceptrons (MLP) on a small dataset. Rahimipour et al. [17] search for number of neurons in two layer MLPs and slopes of the activation functions using GA on a very small (three measurement sites) real-world dataset. A particle swarm optimization algorithm is used in [9] to optimize the amount of neurons in the hidden layers of deep belief networks, learning rate and momentum. However, they also limit their study to a small dataset and fixed the amount of hidden layers. To our knowledge, Pan et al. [15] are the first to implement gradient-based NAS for traffic forecasting in their framework called AutoSTG. They are using a cell-based approach of learning one smaller architecture (a cell) and applying it in sequence multiple times to obtain a larger network, similar to [11]. Their operation space is made up of none, identity, temporal convolution and spatial graph convolution. Additionally, they apply meta learning to learn the adjacency matrices for the spatial graph convolutions and kernels for their temporal convolutions.

To tackle some of the problems described above, we apply evolutionary neural architecture search via genetic algorithm for traffic forecasting. In comparison to the cell-based approach of AutoSTG [15], GA discovers complete variable-sized architectures, decreasing the amount of search hyperparameters. To the best of our knowledge, we are the first to apply ENAS on commonly used real world traffic benchmarks. GA can find architectures that are capable of outperforming or keeping up with current state-of-the-art models.

The work at hand is structured as follows; In Section II we define the problem of traffic forecasting and introduce the bilevel optimization problem NAS solves. We then present our architecture search space and the GA used for exploring it. In Section III we describe the traffic benchmarks, metrics and baseline models. We present the results of our method, compare them with the baseline models, and evaluate the impact of dilated convolution and graph convolution on the prediction performance. In Section IV, we outline possible directions for future work before concluding in Section V.

II. Methodology

A. Traffic forecasting

The task of traffic forecasting is to predict future traffic conditions from historical ones observed at measurement sites of a road network. A road network can be defined as a weighted undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$, where $\mathcal{V}$ is a set of vertices or nodes representing the $|\mathcal{V}| = N$ measurement sites in the road network, $\mathcal{E}$ a set of edges indicating the connectivity between measurement sites and $\mathbf{W} \in \mathbb{R}^{N \times N}$ a weighted adjacency matrix, representing the proximity between nodes. We use distance based adjacency matrices in our work, calculated by driving distances between measurement sites.

Then, given a timestamp t, the traffic conditions on the graph $\mathcal{G}$ are denoted by a graph signal $X_t \in \mathbb{R}^{N \times F}$, where $F \in \mathbb{N}$ is the amount of features observed at each measurement site or node. Finally, the goal of traffic forecasting is to learn a function f for predicting future T graph signals on the graph $\mathcal{G}$ from T' historical graph signals:

$$\mathbf{y}_t = [X_{t+1}, ..., X_{t+T}] = f_\theta([X_{t-T'+1}, ..., X_t]; \mathcal{G}) \in \mathbb{R}^{N \times D \times T}$$

Here, $D \in \mathbb{N}$ denotes the amount of features to predict for each measurement site.

B. Neural architecture search

The objective of NAS is to find an optimal architecture A from the space of architectures $\mathcal{A}$ that minimizes the loss function $\mathcal{L}$ on a given dataset $\mathcal{D}$. To be more precise, we want to solve a bilevel optimization problem:

$$A^* = \min_{A \in \mathcal{A}} \mathcal{L}(\theta^*(A), A, \mathcal{D}_{\text{valid}}) \tag{1}$$

$$\text{s.t.} \quad \theta^*(A) = \arg\min_{\theta} \mathcal{L}(\theta, A, \mathcal{D}_{\text{train}}) \tag{2}$$

Here, $\mathcal{D}_{\text{train}} \subset \mathcal{D}$ and $\mathcal{D}_{\text{valid}} \subset \mathcal{D}$ respectively denote the training and validation datasets and θ the network parameters.

In this work, we use genetic algorithm (GA) to solve the optimization problem of NAS ((1) & (2)) by evolving neural architectures $A \in \mathcal{A}$.

1) Architecture search space: Fig. 1 shows the architecture search space of our method. As can be seen, we are not using a cell-based search approach as in [15], but evolve whole architectures with varying number $N \in \mathbb{N}$ of nodes. The nodes are ordered in a sequence, forming a directed acyclic graph. Each edge (i, j), $i, j \in \mathbb{N}$ is associated with an operation $o^{(i,j)}$ from the operation space $\mathcal{O}$ mapping the node $x^{(i)}$ to node $x^{(j)}$. To obtain node $x^{(j)}$ all of its preceding nodes are summed up:

$$x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)}),\ j = 2, ..., N.$$

The node $x^{(1)}$ is the input node and the node $x^{(N)}$ is the output node of the network. We apply a 2D 1x1 convolution to a given input $X_t \in \mathcal{D}$ to obtain node $x^{(1)}$ and another 2D 1x1 convolution to the output node $x^{(N)}$ followed by a fully connected layer (FC) to obtain the final output. The number of input and output channels $n_c^{(i)}$ and $n_c^{(j)}$ for each operation $o^{(i,j)}$ is fixed to $n_c^{(i)} = \min(2^{i+1}, 128)$.

The operation space is inspired by existing approaches [6], [15], which mainly use convolutional operations. In this work, we use the following operations:

- None – zeroes out the input (no connection between nodes).
- Skip connections – since channels differ between nodes, the skip connection employed is a 2D 1x1 convolution. This operation has no mutable parameters.
- Dilated convolution – dilated convolutions increase the receptive field of a neural network exponentially without adding too many parameters [22]. Hence, we use dilated convolutions in the temporal dimension to capture historic traffic patterns. Note that the receptive field only increases exponentially when dilation factors increase by a factor of two with each following layer [22]. To fulfill this, we modify the dilation factors manually after each crossover and mutation operation. Mutable parameters are the dilation factor and kernel size.
- Graph convolution – graph convolution has been shown to capture the spatial dependencies between measurement sites and has therefore increased the performance of neural architectures [2], [6], [15], [21]. The mutable parameter is the kernel size or the degree of the underlying Chebyshev polynomial.

2) Search method: The GA used in this work is summarised in Algorithm 1. In detail, we start by initializing a random population of size $n_p \in \{2n | n \in \mathbb{N}\}$. Every architecture in the search space has equal probability of getting selected. However, we start with architectures of a random number of nodes between 6 and 8. During evolution there is no upper bound on the network size. To estimate their performance or fitness (MAE), the architectures are trained for a small amount of epochs (see Section III-B) on the training set $\mathcal{D}_{\text{train}}$ and evaluated on the validation set $\mathcal{D}_{\text{valid}}$.

Algorithm 1 Genetic Algorithm

Require: $n_p > 0, n_g > 0$
$population \leftarrow \varnothing$
$best \leftarrow \varnothing$
while $|population| < n_p$ **do**
  $model.arch \leftarrow$ RandomInit()
  $model.fitness \leftarrow$ TrainAndEval($model.architecture$)
  add $model$ to $population$
end while
$c = 0$
while $c < n_g$ **do**
  $offspring \leftarrow \varnothing$
  while $|offspring| < n_p$ **do**
    $parents \leftarrow$ BinaryTournament($population, 2$)
    $children \leftarrow$ UniformCrossover($parents$)
    add $children$ to $offspring$
  end while
  for $model$ in $offspring$ **do**
    $model.arch \leftarrow$ Mutate($model.arch$)
    $model.fitness \leftarrow$ TrainAndEval($model.arch$)
    if $model.fitness < best.fitness$ **then**
      $best \leftarrow model$
    end if
  end for
  add $offspring$ to $population$
  $population \leftarrow$ BinaryTournament($population, n_p$)
  $c += 1$
end while
return $best$

Once the population is initialized and evaluated, the crossover and mutation cycle is repeated $n_g \in \mathbb{N}$ times. We do not use a stopping criterion, but have a fixed amount of cycles. We use binary tournament for selecting two parents for crossover. In binary tournament, two chromosomes are picked at random and the one with better fitness, in our case the MAE on the validation set, gets selected. Hence, better performing architectures are more likely to be selected, but we still retain diversity. After two parents are selected, we apply uniform crossover by selecting a random subset of nodes in the two parent's architectures to be switched. If the sizes of the architectures are different, we switch a maximum of nodes equal to the amount of nodes in the smaller architecture and retain the sizes. The two resulting children are mutated afterwards and added to the current generation. Mutation operations include:

- Switching edges ($o^{(i,j)} \leftrightarrow o^{(i',j')}$)
- Removing an operation ($o^{(i,j)} \leftarrow$ None)
- Changing the type of operation ($o^{(i,j)} \leftarrow o'^{(i,j)}$, where $o' \in \mathcal{O}$ is selected at random)
- Mutating the parameters of an operation (kernel size and/or dilation factor can be increased or decreased to next possible size)

- Adding or removing a node (adding also adds an operation from the operation space, except None)

After mutation, the channels and dilation factors of some operations need to be modified as previously mentioned. All children are then trained and evaluated. Lastly, we use binary tournament to select n_p models from the current generation to stay in the population.

After n_g cycles, we return the model with the best fitness over all generations.

III. Evaluation

A. Experimental settings

1) Datasets: Our experiments are conducted on four real world datasets, two of which are concerned with traffic flow prediction and two with traffic speed prediction:

- PeMSD4 – The PeMSD4 dataset is made up of traffic flow measurements from 307 loop detectors in the San Francisco Bay Area within the period from 1/Jan/2018 - 28/Feb/2018 [2].
- PeMSD8 – The PeMSD8 dataset contains traffic flow measurements from 170 loop detectors in the San Bernardino Area from 1/Jul/2016 to 31/Aug/2016 [2].
- METR-LA – The METR-LA dataset includes traffic speed readings at 207 sensors located on the highways of Los Angeles County from 1/Mar/2012 to 30/Jun/2012 [10].
- PEMS-BAY – The PEMS-BAY dataset comprises traffic speed data from 325 measurement sites in the Bay Area of California from 1/Jan/2017 - 31/May/2017 [10].

All datasets are aggregated into 5 min windows, resulting in 288 timestamps per day. For training, the data is normalized by standard normalization for each node and feature. Given a timestamp t, we want to predict the next hour of traffic conditions, i.e. 12 timesteps. The input $\mathbf{X}_t \in \mathbb{R}^{N\times F\times 12}$ to our network is made up of a recent, daily, and weekly segment from the historical data. These segments are defined as follows:

$$\mathbf{X}_t^{\text{recent}} = [X_{t-11}, ..., X_t]$$
$$\mathbf{X}_t^{\text{daily}} = [X_{t+1-288}, ..., X_{t+12-288}]$$
$$\mathbf{X}_t^{\text{weekly}} = [X_{t+1-7\times 288}, ..., X_{t+12-7\times 288}]$$

As can be seen, the recent segment comprises the last hour of data, the daily segment includes data from the same hour to be predicted, but on the day before and the weekly segment contains the same hour we predict, but one week earlier. Additionally, we include data about time of the day and day of the week for the prediction segment. The segments and time information are stacked in the feature dimension of the input, i.e. $\mathbf{X}_t \in \mathbb{R}^{N\times 5\times 12}$. Stacking in the feature dimension has not been done in previous works. In [7] and [6] multiple modules with the same architectures and a fusion layer are used, while in [2] and [15] only the recent segment is used. We have conducted experiments comparing different input methods and concluded that stacking multiple segments in the feature dimension works best for our approach. However, future research might be conducted to set a standard for the task of traffic prediction.

We seperate the datasets into training, validation and test sets with a 7-1-2 ratio. The adjacency matrices are constructed as in previous works by road network distance and gaussian kernel threshholding [21].

We remark that, due to the choice of inputs, the resulting datasets include fewer samples than in other works, where only the last 12 timesteps are used as inputs [2], [15], [21]. Therefore, direct comparisons with their results are to be taken with caution. For fair comparison in this work, we evaluate the baseline models on our datasets.

2) Metrics: We use mean absolute error (MAE), rooted mean squared error (RMSE) and mean absolute percentage error (MAPE) to evaluate our framework and the baselines:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|,\ RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2},$$
$$MAPE = 100\% \times \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|/y_i$$

Here, $N, \hat{y}_i$ and y_i respectively refer to the number of samples, predicted values and ground truth values. Since y_i can be zero-valued for some measurements, we only compute MAPE when ground truth is larger than one.

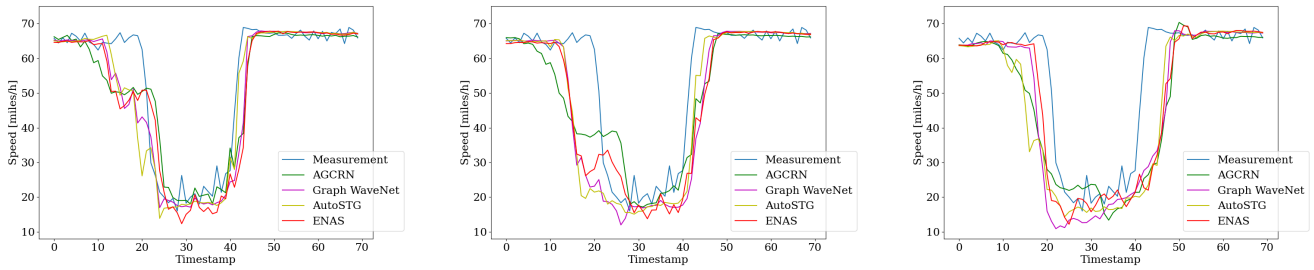
3) Baselines: We compare our framework against the following models:

- Historical average (HA) – Traffic is modeled as a seasonal process. We predict future timesteps by taking the average over the last n_d (to be determined) days of the same time.
- AGCRN – Adaptive graph convolutional recurrent network captures spatio and temporal dependencies automatically from the data without the need of predefined adjacency matrices for the graph convolution [2].
- Graph WaveNet – Deploys WaveNet [19] and graph convolutions for modelling spatio-temporal graph signals. The adjacency matrix is self-adapting by discovering structures in the data without prior knowledge [21].
- AutoSTG – Gradient-based NAS framework for spatio-temporal prediction. Pan et al. [15] use special modules for capturing spatio-temporal dependencies from meta data of the attributed graph.

As mentioned earlier, we evaluate all baselines on our own dataset as described in Section III-A1. To this end, we adapt the publicly available code and conduct the recommended hyperparameter search of each model.

B. Framework settings

We apply GA on each of the four datasets for 100 populations with a starting size of 12 models. The crossover probability and mutation probability are respectively set to 0.9 and 0.075. Note that these have not been fine-tuned, due to long computation times of each experiment. Each of the operations in the operation space (see Section II-B1), except for None,



(a) 15 min horizon. (b) 30 min horizon. (c) 60 min horizon.

Fig. 2: Traffic speed forecast on the METR-LA dataset for a single measurement site for (a) 15 min (b) 30 min and (c) 60 min prediction horizons. Our framework is shown in red (ENAS).

is followed by a ReLU activation and layer normalization [1]. Each network is trained with Adam optimizer for 30 epochs with batch size 64 and a starting learning rate of 0.01, which is reduced to 0 until epoch 30 by a cosine annealing learning rate schedule [13]. The used loss function is MAE as described in Section III-A2. We conduct each experiment on four Nvidia GeForce GTX 3090 GPU to train four models simultaneously. The best model architecture is trained for 100 epochs for different starting learning rates (0.01, 0.005, 0.001) and batch sizes (32, 64, 128). Finally, the best performing model on the validation set is used for measuring performance on the test set.

C. Results and discussion

Table I shows the prediction performance on the four described datasets. We report MAE, RMSE and MAPE for the 15 min, 30 min and 60 min horizons respectively. All experiments are run twice with different random seeds.

As expected, the simplest model HA performs worst on all four data sets. HA cannot capture the required complexity of the spatio-temporal data at hand. It can only model the general trend of the data, but cannot adapt to local changes in the trend.

In contrast, the two hand-crafted deep learning models, AGCRN and Graph WaveNet, can model spatio-temporal dependencies and therefore have better predictive performance. AGCRN lacks stability on the two traffic flow datasets and overall performs worst of all deep learning models. Graph WaveNet outperforms all other methods on the METR-LA dataset. Additionally their performance on PEMS-BAY is best or at least can keep up with AutoSTG and our framework.

We are not able to conduct experiments with AutoSTG on PeMSD4 and PeMSD8, since we lack meta data for these two datasets. AutoSTG [15] is the best performing method on PEMS-BAY for the 15 min horizon. They can also keep up with the other approaches, however lack stability when using different random seeds.

Our framework (ENAS) is the only deep learning model of the four without an adapting adjacency matrix, nevertheless that does not diminish the predictive performance. Especially on PeMSD4, our framework outperforms AGCRN and Graph WaveNet with respect to all metrics and horizons. On PeMSD8 ENAS outperforms on most metrics and horizons. Furthermore, the performance on METR-LA is underwhelming compared to GWN. For PEMS-BAY, our framework can keep up with GWN and AutoSTG.

In terms of search time, our method performs worst as it runs for approximately 300 GPU hours on the smallest dataset (PeMSD8) and approximately 1200 GPU hours on the largest dataset (PEMS-BAY). AutoSTG has to be run multiple times for different combinations of architecture related hyperparameters, which each takes around 10 GPU hours for search and 5 hours for the training of the discovered architectures. AGCRN and Graph WaveNet take the least time, since they do not include an architecture search process.

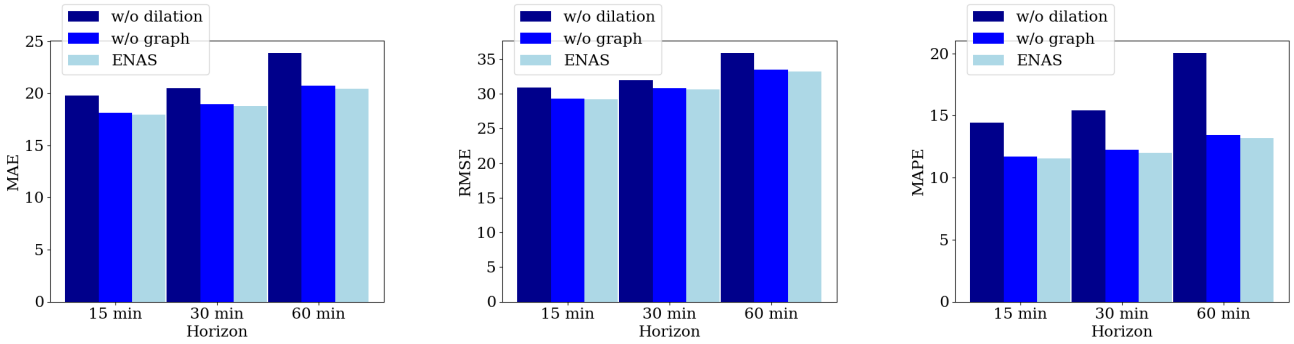
In Fig. 2 we show predictions for a small timewindow of a single measurement site of the METR-LA dataset of all baselines except HA. It can be seen that for a horizon of 15 min and 30 min all investigated models' predictions are similar. Every model detects the future drop in traffic speed prematurely. For the 60 min horizon, Graph WaveNet and our model (ENAS) are closer to the time of the drop, however, overestimate the severance of it. Note that we have shown one example out of thousands present in the dataset. Since the drops in traffic speed are of great interest for ITS, further investigation of the models predictive performance for such is beneficial.

D. Ablation study

To evaluate the importance of dilated convolution and graph convolution in the architectures, we run an ablation study. For that we remove dilated and graph convolution respectively from the operation space and compare the performance of the discovered architectures to our best architecture on PeMSD4. Fig. 3 shows the results for all metrics for 15 min, 30 min and 60 min horizons. It can be seen, that including both operations (as for ENAS) leads to an increase in performance. While dilated convolution (w/o graph) greatly boosts performance, graph convolution (w/o dilation) has less of an impact. We remark, that the architecture without dilated convolution has less network parameters (110k) than the one without graph convolution (286k). However, we retrained with more channels to get a comparable number of parameters, which did not

TABLE I: Traffic forecast performance on PeMSD4, PeMSD8, METR-LA and PEMS-BAY datasets. Here, GWN and ENAS respectively denote Graph WaveNet and our framework.

	MAE			RMSE			MAPE		
PeMSD4	15 min	30 min	60 min	15 min	30 min	60 min	15 min	30 min	60 min
HA	34.33 ± 0.00	34.33 ± 0.00	34.33 ± 0.00	53.27 ± 0.00	53.27 ± 0.00	53.27 ± 0.00	$24.22\% \pm 0.00\%$	$24.22\% \pm 0.00\%$	$24.22\% \pm 0.00\%$
AGCRN	19.02 ± 0.07	19.88 ± 0.11	21.05 ± 0.36	30.99 ± 0.49	32.66 ± 0.53	34.56 ± 0.12	$12.49\% \pm 0.33\%$	$12.92\% \pm 0.32\%$	$13.92\% \pm 0.25\%$
GWN	18.28 ± 0.04	19.24 ± 0.06	20.95 ± 0.04	29.44 ± 0.09	31.09 ± 0.16	33.83 ± 0.21	$11.98\% \pm 0.03\%$	$12.60\% \pm 0.02\%$	$13.71\% \pm 0.00\%$
ENAS	$\mathbf{17.95 \pm 0.01}$	$\mathbf{18.77 \pm 0.00}$	$\mathbf{20.44 \pm 0.02}$	$\mathbf{29.22 \pm 0.01}$	$\mathbf{30.67 \pm 0.00}$	$\mathbf{33.21 \pm 0.03}$	$\mathbf{11.55\% \pm 0.01\%}$	$\mathbf{12.04\% \pm 0.03\%}$	$\mathbf{13.22\% \pm 0.03\%}$
PeMSD8	15 min	30 min	60 min	15 min	30 min	60 min	15 min	30 min	60 min
HA	31.33 ± 0.00	31.33 ± 0.00	31.33 ± 0.00	48.72 ± 0.00	48.72 ± 0.00	48.72 ± 0.00	$23.50\% \pm 0.00\%$	$23.50\% \pm 0.00\%$	$23.50\% \pm 0.00\%$
AGCRN	13.17 ± 0.08	13.67 ± 0.13	14.88 ± 0.23	22.34 ± 0.12	23.66 ± 0.15	25.67 ± 0.19	$8.46\% \pm 0.08\%$	$8.81\% \pm 0.11\%$	$9.73\% \pm 0.24\%$
GWN	13.69 ± 0.15	14.18 ± 0.08	14.98 ± 0.08	21.90 ± 0.13	$\mathbf{23.18 \pm 0.05}$	$\mathbf{25.03 \pm 0.05}$	$8.81\% \pm 0.15\%$	$9.17\% \pm 0.09\%$	$9.94\% \pm 0.02\%$
ENAS	$\mathbf{13.14 \pm 0.01}$	$\mathbf{13.62 \pm 0.17}$	$\mathbf{14.85 \pm 0.18}$	$\mathbf{21.77 \pm 0.07}$	23.18 ± 0.25	25.27 ± 0.15	$\mathbf{8.33\% \pm 0.07\%}$	$\mathbf{8.68\% \pm 0.14\%}$	$\mathbf{9.64\% \pm 0.07\%}$
METR-LA	15 min	30 min	60 min	15 min	30 min	60 min	15 min	30 min	60 min
HA	13.66 ± 0.00	13.66 ± 0.00	13.66 ± 0.00	21.28 ± 0.00	21.28 ± 0.00	21.28 ± 0.00	$19.82\% \pm 0.00\%$	$19.82\% \pm 0.00\%$	$19.82\% \pm 0.00\%$
AGCRN	3.38 ± 0.00	4.07 ± 0.00	5.04 ± 0.00	7.48 ± 0.00	9.28 ± 0.00	11.34 ± 0.00	$8.46\% \pm 0.00\%$	$10.39\% \pm 0.00\%$	$12.90\% \pm 0.00\%$
GWN	$\mathbf{2.84 \pm 0.01}$	$\mathbf{3.22 \pm 0.01}$	$\mathbf{3.62 \pm 0.04}$	$\mathbf{5.45 \pm 0.03}$	$\mathbf{6.44 \pm 0.00}$	$\mathbf{7.39 \pm 0.05}$	$\mathbf{7.40\% \pm 0.05\%}$	$\mathbf{8.67\% \pm 0.14\%}$	$\mathbf{10.14\% \pm 0.20\%}$
AutoSTG	2.94 ± 0.13	3.40 ± 0.12	3.97 ± 0.14	5.71 ± 0.27	6.81 ± 0.13	8.02 ± 0.06	$7.62\% \pm 0.43\%$	$9.18\% \pm 0.48\%$	$11.09\% \pm 0.60\%$
ENAS	2.97 ± 0.00	3.40 ± 0.01	3.88 ± 0.01	5.75 ± 0.02	6.78 ± 0.01	7.80 ± 0.01	$7.90\% \pm 0.04\%$	$9.50\% \pm 0.07\%$	$11.27\% \pm 0.07\%$
PEMS-BAY	15 min	30 min	60 min	15 min	30 min	60 min	15 min	30 min	60 min
HA	3.28 ± 0.00	3.28 ± 0.00	3.28 ± 0.00	6.54 ± 0.00	6.54 ± 0.00	6.54 ± 0.00	$7.99\% \pm 0.00\%$	$7.99\% \pm 0.00\%$	$7.99\% \pm 0.00\%$
AGCRN	1.41 ± 0.03	1.72 ± 0.01	1.99 ± 0.01	2.95 ± 0.02	3.89 ± 0.01	4.56 ± 0.02	$3.09\% \pm 0.01\%$	$3.99\% \pm 0.00\%$	$4.79\% \pm 0.00\%$
GWN	1.33 ± 0.01	$\mathbf{1.62 \pm 0.02}$	$\mathbf{1.90 \pm 0.02}$	2.81 ± 0.02	3.71 ± 0.04	$\mathbf{4.44 \pm 0.11}$	$2.84\% \pm 0.01\%$	$3.74\% \pm 0.04\%$	$4.59\% \pm 0.12\%$
AutoSTG	$\mathbf{1.31 \pm 0.03}$	1.63 ± 0.07	1.94 ± 0.09	$\mathbf{2.79 \pm 0.09}$	3.78 ± 0.15	4.61 ± 0.22	$\mathbf{2.74\% \pm 0.07\%}$	$\mathbf{3.71\% \pm 0.17\%}$	$4.68\% \pm 0.30\%$
ENAS	1.32 ± 0.00	1.63 ± 0.00	1.91 ± 0.00	2.81 ± 0.00	$\mathbf{3.71 \pm 0.01}$	4.45 ± 0.01	$2.82\% \pm 0.00\%$	$3.74\% \pm 0.01\%$	$\mathbf{4.59\% \pm 0.00\%}$



(a) MAE (b) RMSE (c) MAPE

Fig. 3: Comparison of the prediction performance of our framework on PeMSD4 for different operation spaces. We compare the complete operation space (ENAS), without graph convolution (w/o graph) and without dilated convolution (w/o dilation).

increase the performance drastically. The worse performance of only graph convolutions can be due to the adjacency matrix not incorporating spatial dependencies correctly. Pan et al. [15] have seen the opposite effects of their modules, hence, we conject that learning spatial features from meta data or from data directly is a superior method to ours.

IV. Future work

We have shown that our framework can compete with existing approaches. Nevertheless, there are still a variety of potential improvements that can be implemented. The GA used can be varied in terms of crossover and mutation probabilities, population size, mutation operations, and selection methods. Network morphisms [3], weight inheritance or estimating performance without training [12] can speed up the search and allow larger population sizes. This would be beneficial, since due to the small population size in our study, the search space is not extensively explored, leading to only small performance gains over time. In addition, another evolutionary algorithm such as particle swarm optimization [9] should be compared against our algorithm.

We do not use adaptive adjacency matrices as in the deep learning baselines. In future work, these could be learned from metadata as in AutoSTG [15] or from the graph signal data as in AGCRN [2] and Graph WaveNet [21].

Finally, of particular interest is the structure and amount of input data. While many methods use only the last hour of data as input [2], [15], [21], some, as in our approach, add daily and weekly periodic segments, sometimes dating back not just one but several weeks [6], [7]. Incorporating more historical data for prediction could increase performance, hence a more thorough investigation of this topic is worthwhile.

V. Conclusion

In this work, we apply evolutionary neural architecture search using genetic algorithm for spatio-temporal traffic prediction. Our framework discovers entire architectures at once, consisting of dilated convolutions and graph convolutions, to learn temporal and spatial dependencies, respectively. Experiments with two traffic speed benchmarks and two traffic flow benchmarks demonstrate the effectiveness of our system. In the future, we plan to extend our framework by increasing the complexity of our architectures, automatically adapting the adjacency matrix from the data, and improving our genetic algorithm.

References

[1] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016.

[2] Lei Bai, Lina Yao, Can Li, Xianzhi Wang, and Can Wang. Adaptive graph convolutional recurrent network for traffic forecasting. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[3] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[4] Rui Fu, Zuo Zhang, and Li Li. Using lstm and gru neural network methods for traffic flow prediction. In *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, pages 324–328, 2016.

[5] Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. Graph neural architecture search. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, IJCAI'20, 2021.

[6] Liang Ge, Siyu Li, Yaqian Wang, Feng Chang, and Kunyan Wu. Global spatial-temporal graph convolutional network for urban traffic speed prediction. *Applied Sciences*, 10(4), 2020.

[7] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):922–929, Jul. 2019.

[8] Daniel Klosa, Amin Mallek, and Christof Büskens. Short-term traffic flow forecast using regression analysis and graph convolutional neural networks. In *2021 IEEE 23rd Int Conf on High Performance Computing Communications; 7th Int Conf on Data Science Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud Big Data Systems Application (HPCC/DSS/SmartCity/DependSys)*, pages 1413–1418, 2021.

[9] Linchao Li, Lingqiao Qin, Xu Qu, Jian Zhang, Yonggang Wang, and Bin Ran. Day-ahead traffic flow forecasting based on a deep belief network optimized by the multi-objective particle swarm algorithm. *Knowledge-Based Systems*, 172:1–14, 2019.

[10] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *International Conference on Learning Representations*, 2018.

[11] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

[12] Vasco Lopes, Saeid Alirezazadeh, and Luís A. Alexandre. Epe-nas: Efficient performance estimation without training for neural architecture search. *Artificial Neural Networks and Machine Learning – ICANN 2021*, page 552–563, 2021.

[13] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

[14] Spyros Makridakis and Michele Hibon. Arma models and the box–jenkins methodology. *Journal of Forecasting*, 1997.

[15] Zheyi Pan, Songyu Ke, Xiaodu Yang, Yuxuan Liang, Yong Yu, Junbo Zhang, and Yu Zheng. Autostg: Neural architecture search for predictions of spatio-temporal graph. In *Proceedings of the Web Conference 2021*, WWW '21, page 1846–1855, New York, NY, USA, 2021. Association for Computing Machinery.

[16] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4095–4104. PMLR, 10–15 Jul 2018.

[17] Shiva Rahimipour, Ray Moienfar, and S. Mehdi Hashemi. Traffic prediction using a self-adjusted evolutionary neural network. *Journal of Modern Transportation*, 27, 12 2018.

[18] Hongyu Sun, Henry X. Liu, Heng Xiao, Rachel R. He, and Bin Ran. Use of local linear regression model for short-term traffic forecasting. *Transportation Research Record*, 2003.

[19] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. In *Proc. 9th ISCA Workshop on Speech Synthesis Workshop (SSW 9)*, page 125, 2016.

[20] Eleni I. Vlahogianni, Matthew G. Karlaftis, and John C. Golias. Optimized and meta-optimized neural networks for short-term traffic flow prediction: A genetic approach. *Transportation Research Part C: Emerging Technologies*, 13(3):211–234, 2005.

[21] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. Graph wavenet for deep spatial-temporal graph modeling. In *IJCAI*, 2019.

[22] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *International Conference on Learning Representations (ICLR)*, May 2016.

[23] Kaixiong Zhou, Qingquan Song, Xiao Huang, and Xia Hu. Auto-gnn: Neural architecture search of graph neural networks. *ArXiv*, abs/1909.03184, 2019.

[24] Eric Zivot and Jiahui Wang. *Vector Autoregressive Models for Multivariate Time Series*. Springer New York, 2003.

[25] Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

machine learning & knowledge extraction

MDPI

Article

Low Cost Evolutionary Neural Architecture Search (LENAS) Applied to Traffic Forecasting †

Daniel Klosa * and Christof Büskens

WG Optimization and Optimal Control, Center for Industrial Mathematics, University of Bremen, 28359 Bremen, Germany; bueskens@uni-bremen.de

* Correspondence: dklosa@uni-bremen.de

† This paper is an extended version of our paper published in the Proceedings of the 21st IEEE International Conference on Machine Learning and Applications (ICMLA), Nassau The Bahamas, 12–14 December 2022.

Abstract: Traffic forecasting is an important task for transportation engineering as it helps authorities to plan and control traffic flow, detect congestion, and reduce environmental impact. Deep learning techniques have gained traction in handling such complex datasets, but require expertise in neural architecture engineering, often beyond the scope of traffic management decision-makers. Our study aims to address this challenge by using neural architecture search (NAS) methods. These methods, which simplify neural architecture engineering by discovering task-specific neural architectures, are only recently applied to traffic prediction. We specifically focus on the performance estimation of neural architectures, a computationally demanding sub-problem of NAS, that often hinders the real-world application of these methods. Extending prior work on evolutionary NAS (ENAS), our work evaluates the utility of zero-cost (ZC) proxies, recently emerged cost-effective evaluators of network architectures. These proxies operate without necessitating training, thereby circumventing the computational bottleneck, albeit at a slight cost to accuracy. Our findings indicate that, when integrated into the ENAS framework, ZC proxies can accelerate the search process by two orders of magnitude at a small cost of accuracy. These results establish the viability of ZC proxies as a practical solution to accelerate NAS methods while maintaining model accuracy. Our research contributes to the domain by showcasing how ZC proxies can enhance the accessibility and usability of NAS methods for traffic forecasting, despite potential limitations in neural architecture engineering expertise. This novel approach significantly aids in the efficient application of deep learning techniques in real-world traffic management scenarios.

Keywords: neural architecture search; traffic forecasting; zero-cost proxies

Citation: Klosa, D.; Büskens, C. Low Cost Evolutionary Neural Architecture Search (LENAS) Applied to Traffic Forecasting. *Mach. Learn. Knowl. Extr.* **2023**, *5*, 830–846. https://doi.org/10.3390/make5030044

Academic Editors: Moamar Sayed-Mouchaweh, Mohd Arif Wani, Vasile Palade and Mehmed M. Kantardzic

Received: 13 June 2023
Revised: 17 July 2023
Accepted: 24 July 2023
Published: 28 July 2023

1. Introduction

Forecasting future traffic conditions, such as flow and speed, by analyzing historical traffic patterns is an essential task for transportation engineering. Accurate traffic forecasts can detect congestion and help authorities plan and control traffic flow, enabling intelligent traffic systems (ITS) to adjust to future events, leading to more uniform traffic flow, and reduced CO_2 emissions, ultimately reducing environmental impact. With the increasing availability of traffic data, there has been a growing interest in developing machine learning algorithms for traffic forecasting.

However, traffic forecasting poses several challenges. Future traffic conditions at a single measurement site depend not only on recent conditions in the temporal dimension but also on upstream measurements in the spatial dimension. Additionally, large datasets with hundreds of measurement sites and complex road networks can make modeling dependencies between them difficult.

Linear regression [1], auto-regressive moving average [2], vector auto-regression [3], and k-nearest neighbors [4,5] are traditional methods that fall short in capturing the complex

spatio-temporal dependencies present in large traffic datasets. As a result, recent research has shifted towards deep learning models such as long-short term memory and gated recurrent unit [6] to capture temporal dependencies, and graph convolutional neural networks (GCN) to learn spatial dependencies within the data [7–9].

One limitation of GCN models is their reliance on prior knowledge of the spatial connections within the graph-structured road network, typically in the form of an adjacency matrix. GraphWaveNet [7] and AGCRN [10] address this issue by learning spatial dependencies directly from the data. However, these methods still rely on handcrafted neural architectures designed by experts. Moreover, deploying these approaches in real-world applications requires additional customization for the specific scenario, which can be time-consuming and require considerable effort.

Neural architecture search (NAS) methods have gained popularity in recent years for discovering customized neural architectures for various tasks, reducing the tedious process of neural architecture design. While early NAS frameworks focused on computer vision and language modeling [11,12], they can also be applied to graph data [13,14] and spatio-temporal data [15].

NAS typically involves three components: the search space, search strategies, and performance estimation. The search space defines the general structure of discovered network architectures, including the operations and their connections within the network. Different search strategies exist, such as reinforcement learning (RL), gradient-based search, and evolutionary NAS (ENAS). RL-based algorithms are known to require significant computational resources, even on smaller datasets such as CIFAR-10 [16]. Gradient-based NAS methods, such as those used in [12], are more efficient but can become trapped in local minima and require the construction of a supernet in advance. ENAS can explore the search space more thoroughly without a given supernet, but can also suffer from long computation times. The performance estimation strategy defines how discovered architectures are evaluated. Typically, this involves training them for a certain number of epochs, which can be costly. However, techniques such as weight inheritance and network morphisms eliminate the need for training from scratch, greatly reducing training time [17]. It is also possible to evaluate discovered architectures without training at all [18,19].

The research on NAS for traffic forecasting is limited. Early work [20] investigates the implementation of genetic algorithm (GA) for optimizing gradient descent hyperparameters and hidden layers in multi layer perceptrons (MLP) on a small dataset. Rahimipour et al. [21] search for number of neurons in two layer MLPs and slopes of the activation functions using GA on a small (three measurement sites) real-world dataset. A particle swarm optimization algorithm is used in [22] to optimize the amount of neurons in the hidden layers of deep belief networks, learning rate and momentum. However, they also limit their study to a small dataset and fix the amount of hidden layers. To our knowledge, Pan et al. [15] are the first to implement gradient-based NAS for traffic forecasting in their framework called AutoSTG. They are using a cell-based approach of learning one smaller architecture (a cell) and applying it in sequence multiple times to obtain a larger network, similar to [12]. Their operation space is made up of none, identity, temporal convolution, and spatial graph convolution. Additionally, they apply meta learning to learn the adjacency matrices for the spatial graph convolutions and kernels for their temporal convolutions. To our knowledge, Klosa and Büskens [23] are the first to apply ENAS for the task of traffic forecasting on four real world datasets. They use a simple genetic algorithm to search through an architecture space flexible in size. Their algorithm does not make use of performance estimation strategies, which leads to tremendous computation times, rendering their approach unfit for application in the real world.

Our study proposes to advance the field by integrating zero-cost (ZC) proxies into the architecture search algorithm used by Klosa and Büskens [23]. ZC proxies offer the advantage of being able to rank neural architectures within the search space without necessitating expensive training [18,19]. This technique has shown promising results in image classification, natural language processing, and computer vision. Our proposed

application of ZC proxies to spatio-temporal data forecasting and specifically to traffic data is therefore novel.

However, it is crucial to acknowledge the potential limitations of this approach. ZC proxies have predominantly been tested in fields other than regression tasks, and their effectiveness in traffic forecasting is not yet fully understood. Additionally, some potential challenges may arise in terms of biases towards architecture size, stability with regard to weight initialization and mini-batch sampling, and the correlation between ZC proxies and validation loss.

To rigorously investigate these issues, our research will aim to answer the following questions:

1. Are ZC proxies biased towards architecture size?
2. Are ZC proxies stable with regards to weight initialization and mini-batch sampling?
3. Are ZC proxies stable with regards to architecture size?
4. Are ZC proxies and validation loss correlated?

Addressing these questions will provide a deeper understanding of the capabilities and limitations of ZC proxies in the context of traffic forecasting, ultimately helping to determine whether this method can be applied reliably in real-world settings.

This research paper is structured as follows; In Section 2 we define the problem of traffic forecasting, introduce the bilevel optimization problem NAS solves, the architecture search space, the genetic algorithm and define the ZC proxies examined in this work. We answer the above mentioned research questions in Section 2.6. We describe the experimental setup of our low cost evolutionary neural architecture search (LENAS) in Section 2.7 and evaluate the performance on four real world datasets in Section 3. In Section 4, we discuss the results, outline possible directions for future work before comming to a conclusion.

2. Materials and Methods

In this section, we describe the task of traffic forecasting. Afterwards, we define neural architecture search and the components making up our framework LENAS. For that, we define the search space, the search method and ZC proxies as our performance estimation. Afterwards we answer the above stated research questions and describe the experimental setup of the LENAS framework.

2.1. Traffic Forecasting

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$ denote an undirected graph, where $\mathcal{V}$ is a set of vertices or nodes representing the $\mid \mathcal{V} \mid = N$ measurement sites in the road network, $\mathcal{E}$ a set of edges indicating the connectivity between measurement sites and $\mathbf{W} \in \mathbb{R}^{N \times N}$ a weighted adjacency matrix, representing the proximity between nodes. Then, given a timestamp t, the traffic conditions on the graph $\mathcal{G}$ are denoted by a graph signal $X_t \in \mathbb{R}^{N \times F}$, where $F \in \mathbb{N}$ is the amount of features observed at each measurement site or node. Finally, the goal of traffic forecasting is to learn a function f for predicting future T graph signals on the graph $\mathcal{G}$ from T' historical graph signals:

$$\mathbf{y}_t = [X_{t+1}, ..., X_{t+T}] = f_\theta([X_{t-T'+1}, ..., X_t]; \mathcal{G}) \in \mathbb{R}^{N \times D \times T}$$

Here, $D \in \mathbb{N}$ denotes the amount of features to predict for each measurement site.

2.2. Neural Architecture Search

The objective of NAS is to find an optimal architecture A from the space of architectures $\mathcal{A}$ that minimizes the loss function $\mathcal{L}$ on a given dataset $\mathcal{D}$. To be more precise, we want to solve a bilevel optimization problem:

$$A^* = \min_{A \in \mathcal{A}} \mathcal{L}(\theta^*(A), A, \mathcal{D}_{\text{valid}}) \tag{1}$$

$$\text{s.t.} \quad \theta^*(A) = \arg\min_{\theta} \mathcal{L}(\theta, A, \mathcal{D}_{\text{train}}) \tag{2}$$

Here, $\mathcal{D}_{\text{train}} \subset \mathcal{D}$ and $\mathcal{D}_{\text{valid}} \subset \mathcal{D}$ respectively denote the training and validation datasets and θ the network parameters.

To solve the bilevel optimization problem, NAS can be split into three components: the architecture search space, the search method, and the performance estimation, which are described in the following sections.

2.3. Architecture Search Space

Figure 1 shows the architecture search space of our method. As can be seen, we are not using a cell-based search approach as in [15], but evolve whole architectures with varying number $N \in \mathbb{N}$ of nodes. The nodes are ordered in a sequence, forming a directed acyclic graph. Each edge (i, j), $i, j \in \mathbb{N}$ is associated with an operation $o^{(i,j)}$ from the operation space $\mathcal{O}$ mapping the node $x^{(i)}$ to node $x^{(j)}$. To obtain node $x^{(j)}$ all of its preceding nodes are summed up:

$$x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)}), \; j = 2, ..., N.$$

The node $x^{(1)}$ is the input node and the node $x^{(N)}$ is the output node of the network. We apply a 2D 1×1 convolution to a given input $X_t \in \mathcal{D}$ to obtain node $x^{(1)}$ and another 2D 1×1 convolution to the output node $x^{(N)}$ followed by a fully connected layer (FC) to obtain the final output. The number of input and output channels $n_c^{(i)}$ and $n_c^{(j)}$ for each operation $o^{(i,j)}$ is fixed to $n_c^{(i)} = \min(2^{i+1}, 128)$.

The operation space is inspired by existing approaches [8,15], which mainly use convolutional operations. In this work, we use the following operations:

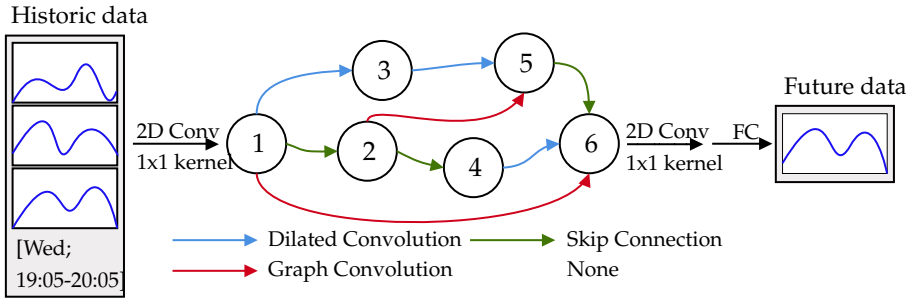


Figure 1. Architecture search space of our framework.

2.3.1. None

The none or zero operation zeroes out the input. This is equivalent to not having a connection between nodes.

2.3.2. Skip Connection

Skip connection usually copies the input. However, since channels differ between nodes, the skip connection employed is a 2D 1×1 convolution that upscales the channel dimension when necessary. This operation has no mutable parameters.

2.3.3. Dilated Causal Convolution

Dilated convolutions are a modification of the standard convolutional operations, designed to increase the receptive field of a network without substantially increasing the number of parameters or the computational cost [24]. Specifically, dilated causal convolutions represent a type of convolution that only allows access to past (causal) information, a feature that is particularly useful when processing sequential or temporal data such as in the cases of time-series prediction or speech synthesis.

In the standard convolutional operations, the elements of the filter are applied to the input elements in a compact, contiguous manner. On the contrary, in dilated convolutions,

the filter is applied to the input with gaps, which are determined by a dilation rate. This leads to an exponential expansion of the receptive field as the size of the filter grows linearly. This combination of causality with the increased receptive field enables the network to efficiently capture long-range temporal dependencies.

Note that the receptive field only increases exponentially when dilation factors increase by a factor of two with each following layer [24]. To fulfill this, we modify the dilation factors manually after each crossover and mutation operation. Mutable parameters are the dilation factor and kernel size.

2.3.4. Graph Convolution

Graph convolutional networks (GCNs) have displayed significant effectiveness in various applications owing to their ability to capture topological correlations present in graph-structured data [7,8,10,15]. Among several methodologies to perform graph convolutions, the method of Chebyshev polynomial approximation has been particularly notable.

The graph convolution operation based on Chebyshev polynomial allows the network to take into account different scales of neighborhood when processing a node in the graph. From the perspective of spectral graph theory, the Chebyshev polynomial approximation has been used to generalize the convolution operation in the Fourier domain, leading to computational efficiency and ensuring a flexible receptive field.

The central concept is to approximate the spectral decomposition of the graph Laplacian, a crucial element of graph Fourier transform, with Chebyshev polynomials. Given a signal x on a graph and a filter defined as a function g_θ of the Laplacian L, the convolution of x with g_θ on the graph is represented in the spectral domain:

$$g_\theta * x = g_\theta(L)x$$

To circumvent the computational overhead associated with the spectral decomposition of the Laplacian, especially for large graphs, the filter g_θ can be approximated using Chebyshev polynomials T_k:

$$g_\theta \approx \sum_{k=0}^{K} \theta_k T_k(\tilde{L})$$

where $\tilde{L} = \frac{2L}{\lambda_{\max}} - I$ is the scaled Laplacian, $\lambda_{\max}$ is the largest eigenvalue of L, and I is the identity matrix. T_k can be computed recursively as:

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$$

with $T_0(x) = 1$ and $T_1(x) = x$. Consequently, the filter becomes a K-localized operator, meaning it relies only on the K-hop neighborhood of each node, where K is the order of the polynomial. This method leads to a significant reduction in computational complexity while providing control over the balance between the model's capacity and computational efficiency. K is the mutable parameter in this operation.

2.4. Search Method

We use the same genetic algorithm (GA) as a search method for NAS as in our previous work [11] with the addition of using ZC proxies as performance estimators. The GA is summarised in Algorithm 1. We warmstart the genetic algorithm by choosing a large starting population size n_w and selecting the best n_p performing architectures for the following n_c cycles. For performance estimation of each architecture we use the naswot ZC proxy as described in Section 2.5 due to the performance in the experimental Section 2.6.

Algorithm 1 Genetic algorithm with naswot

Require: $n_w > 0, n_p > 0, n_c > 0$
$population \leftarrow \varnothing$
$best \leftarrow \varnothing$
while $|population| < n_w$ **do** ▷ Warmstart
  $model.arch \leftarrow$ RandomInit()
  $model.fitness \leftarrow$ naswot($model.architecture$)
  add *model* to *population*
end while
$population \leftarrow$ Elitism($population$) ▷ best genomes
for i in $range(n_c)$ **do**
  $offspring \leftarrow \varnothing$
  while $| offspring | < n_p$ **do**
    $parents \leftarrow$ BinaryTournament($population, 2$)
    $children \leftarrow$ UniformCrossover($parents$)
    add *children* to $offspring$
  end while
  for *model* in $offspring$ **do**
    $model.arch \leftarrow$ Mutate($model.architecture$)
    $model.fitness \leftarrow$ naswot($model.architecture$)
    if $model.fitness < best.fitness$ **then**
      $best \leftarrow model$
    end if
  end for
  add $offspring$ to *population*
  $population \leftarrow$ BinaryTournament($population, n_p$)
end for
return *best*

Once the population is initialized and evaluated, the crossover and mutation cycle is repeated $n_g \in \mathbb{N}$ times. We do not use a stopping criterion, but have a fixed amount of cycles. We use binary tournament for selecting two parents for crossover. In binary tournament, two chromosomes are picked at random and the one with better fitness, in our case the ZC proxy score, is selected. Hence, better performing architectures are more likely to be selected, but we still retain diversity. After two parents are selected, we apply uniform crossover by selecting a random subset of nodes in the two parent's architectures to be switched. If the sizes of the architectures are different, we switch a maximum of nodes equal to the amount of nodes in the smaller architecture and retain the sizes. Then, the two resulting children are mutated and added to the current generation. Mutation operations include:

- Switching edges ($o^{(i,j)} \leftrightarrow o^{(i',j')}$)
- Removing an operation ($o^{(i,j)} \leftarrow$ None)
- Changing the type of operation ($o^{(i,j)} \leftarrow o'^{(i,j)}$, where $o' \in \mathcal{O}$ is selected at random)
- Mutating the parameters of an operation (kernel size and/or dilation factor can be increased or decreased to next possible size)
- Adding or removing a node (adding also adds an operation from the operation space, except None)

After mutation, the channels and dilation factors of some operations need to be modified as previously mentioned. All children are then trained and evaluated. Lastly, we use a binary tournament to select n_p models from the current generation to stay in the population.

After n_c cycles, we return the model with the best fitness (naswot score) over all generations.

2.5. Zero-Cost Proxies

As mentioned earlier, performance estimation is the bottleneck of NAS when it comes to computation time. Zero-cost (ZC) proxies aim at evaluating an architectures performance without the need of training, i.e., they evaluate network performance from one forward pass and/or backwards pass of a single mini-batch of random data. In the following, we give a brief overview of the zero-cost proxies used in this work. The ZC proxies snip, grasp, synflow, and Fisher are inspired by pruning theory in which they are used to prune network parameters least contributing to network performance. In recent works they have been applied to score whole neural networks without training [19,25]. The ZC proxies jacob_cov and naswot have been solely designed with scoring networks for NAS in mind. We note that all ZC proxies have been thoroughly investigated for classification tasks [19,25], however, research on regression is to the authors' knowledge non-existent.

2.5.1. Gradient Norm

In gradient norm (grad_norm) the Euclidean norm of the gradients resulting from one mini-batch of data is summed up. Ref. [25] use it in their work on ZC proxies as a baseline.

2.5.2. Single-Shot Network Pruning

Single-shot network pruning (snip) was proposed in [26] for parameter pruning at initialisation stage of neural networks. It was used in [25] as a ZC proxy by computing

$$snip(\theta) = |\frac{\partial \mathcal{L}}{\partial \theta} \odot \theta|$$

for each parameter θ in the architecture A and obtaining the sum $snip(A) = \sum_{\theta \in A} snip(\theta)$. $\odot$ denotes the Hadamard product.

2.5.3. Gradient Signal Preservation

Gradient signal preservation (grasp) was introduced to improve upon snip in [27]. The idea being to incorporate the change of gradient norm instead of loss when pruning a parameter:

$$grasp(\theta) = -\left(H\frac{\partial \mathcal{L}}{\partial \theta}\right) \odot \theta$$

Here, H denotes the Hessian. It was used in [25] as a ZC proxy by computing the sum $grasp(A) = \sum_{\theta \in A} grasp(\theta)$.

2.5.4. Synaptic Flow Pruning

Synaptic flow pruning (synflow) has been introduced as a method of pruning network parameters without the need of training or data [28]. It does so by taking the product of all network parameters as a loss $\mathcal{R}$, avoiding layer collapse:

$$synflow(\theta) = \frac{\partial \mathcal{R}}{\partial \theta} \odot \theta$$

It was used in [25] as a ZC proxy by computing the sum $synflow(A) = \sum_{\theta \in A} synflow(\theta)$.

2.5.5. Fisher

Fisher was introduced in [29] as a method to prune activation channels having the least effect on the loss in a neural network. It computes the sum over all gradients of the activation layers a in an architecture:

$$fisher(a) = \left(\frac{\partial \mathcal{L}}{\partial a} a\right)^2$$

To use it as a ZC proxy, we compute the sum $fisher(A) = \sum_{a \in A} fisher(a)$ as in [25].

2.5.6. Jacob Covariance

Jacobian covariance (jacob_cov) as introduced in [30] measures the flexibility of a neural network by computing the covariance of the Jacobians of the rectified linear units for a minibatch. The idea being that for a network to be able to tell inputs apart the covariance should be low. For more details we refer to the original work [30].

2.5.7. NAS without Training

NAS without training (naswot) is what we will call the successor of jacob_cov as described in [30]. Naswot builds on the same idea, but instead of computing the covariance of the Jacobians, it computes a distance metric based on the activations of the rectified linear units within a network. Given a minibatch $X = \{x_i\}_{i=1}^{b}$ of size $b \in \mathbb{N}$. After a forward pass of the minibatch we obtain the activations of each rectified linear unit a_i. Then, the activations are flattened and converted into a binary code c_i, s.t. $c_{i,m} = 0$ if $a_{i,m} = 0$ and $c_{i,m} = 1$ if $a_{i,m} > 0$. Afterwards, we compute the Hamming distance $d_H(c_i, c_j) \in [0, 1]$ of the binary codes to measure their similarity. We then obtain the matrix

$$K_H = \begin{pmatrix} d_H(c_1, c_1) & \cdots & d_H(c_1, c_b) \\ \vdots & & \\ d_H(c_b, c_1) & \cdots & d_H(c_b, c_b) \end{pmatrix}$$

and finally compute the naswot score:

$$naswot(A) = \log|K_H|$$

We use the implementation of [25] for all ZC proxies except naswot, where we used the implementation as in [30] and made some adjustments to make it viable to use for regression tasks instead of classification as intended by the authors.

2.6. Robustness, Bias, and Usability of ZC Proxies

In this section, we aim to answer the research questions with regards to robustness, bias, and usability of zero-cost proxies for NAS in our setting. All of the following experiments are carried out on the PeMSD8 dataset described in Section 2.7.

Since we want to use ZC proxies in genetic algorithm as a measure of fitness, the resulting scores and true fitness should lead to the same or similar ranking within the population. Hence, in the following experiments, we will sample a population with different architectures from our search space and compute the Spearman rank correlation of ZC proxy score and true fitness. The true fitness is determined by training the architectures until they converge and taking the best validation loss. As a loss function, we use the MAE for training.

The main objective is to find a ZC proxy with high correlation to the validation loss. However, there are also questions to be answered when it comes to robustness. We want to find a ZC proxy that is not affected by the weight initialization of the network, nor the sampling of the mini batch used for computation. Furthermore, the choice of channel depths and the size of the mini batch should not affect the score. Previous work [19] has discussed that the size of the architecture, i.e., the amount of layers in a network might have an effect on ZC proxy scoring. To examine this behaviour, apart from the ZC proxy score z, we will also compute

$$z_l = z/n_l, \qquad z_c = z/n_c$$

as the scaled variants of the score by the number of layers n_l and number of channels n_c in each network. The results will answer research question 1 as stated in the Introduction.

2.6.1. Are ZC Proxies Robust with Regards to Weight Initialization and Mini-Batch Sampling?

To answer the second research question, we compute ZC proxy scores for each of 30 sampled architectures on 24 different random seeds. We obtain a ranking of architectures by score for each random seed. Then, we compare the rankings by computing the Spearman rank correlation. Correlation close to 0 indicates that rankings are not correlated, rendering the ZC proxy unusable. Correlation close to 1 (or -1 when negative correlation) indicates similar or same rankings. Additionally, we repeat this process for three different sizes of mini batches (24, 32 and 64). We show the results in Table 1, where we take the mean Spearman rank correlation over the three mini batch sizes.

It can be seen that grad_norm, snip, and synflow obtain good correlations, while naswot performs the best. Scaling by number of layers and number of channels greatly improves the jacob_cov score and improves naswot to reach a perfect correlation. Scaling synflow improves the proxy slightly and grad_norm, snip, grasp, and Fisher get worse when scaled.

According to these results, naswot is the best choice when scaled since the Spearman rank correlation is perfect. It does not matter which random seed or mini-batch sampling is chosen, naswot scored the architectures in the same order every time.

Table 1. Mean Spearman rank correlation over multiple random seeds for each ZC proxy score z and its scaled variants by layers z_l and channels z_c.

	Grad_norm	**Snip**	**Grasp**	**Fisher**	**Synflow**	**Jacob_cov**	**Naswot**
z	0.739	0.809	0.491	0.321	0.793	0.473	**0.917**
z_l	0.395	0.454	0.308	0.209	0.824	0.983	**0.999**
z_c	0.605	0.535	0.377	0.217	0.841	0.987	**0.999**

2.6.2. Are ZC Proxies Robust with Regards to Architecture Size?

For the third research question, we compute ZC proxy scores of the 30 sampled architectures for different size configurations. We initialize each of the 30 architectures with the channel depth at first layer $c_1 \in \{4, 8\}$ and maximum channel depth throughout the architecture $c_{\max} \in \{32, 64, 128\}$, resulting in six different combinations. For each of the six size combination, we score the 30 architectures and rank them accordingly. Afterwards we compute the Spearman rank correlation between these rankings. Additionally, this experiment is repeated for 24 different random seeds. The results are shown in Table 2, where we show the mean Spearman rank correlation over the 6 hyperparameter combinations and 24 random seeds.

Again, scaled naswot shows the most robustness closely followed by scaled jacob_cov, hence, the choice of hyperparameters does not matter when using these two ZC proxies. All other ZC proxies are not robust with respect to hyperparameter choice, and therefore, if used, hyperparameters need to be chosen carefully when using these ZC proxies. We note that these results are also affected by the robustness of ZC proxies with respect to the initialization, i.e., low correlation between random seeds also results in low correlation with respect to hyperparameter choice.

Table 2. Mean Spearman rank correlation over multiple architecture sizes for each ZC proxy score z and its scaled variants by layers z_l and channels z_c.

	Grad_norm	**Snip**	**Grasp**	**Fisher**	**Synflow**	**Jacob_cov**	**Naswot**
z	0.741	0.809	0.492	0.325	0.787	0.475	**0.908**
z_l	0.371	0.361	0.296	0.213	0.822	0.983	**0.999**
z_c	0.773	0.707	0.520	0.258	0.857	0.990	**0.997**

2.6.3. Are ZC Proxies and Validation Loss Correlated?

To answer the fourth research question, we sample 161 random architectures from our search space. As for the third research question, we initialize each of the 161 architectures with the channel depth at first layer $c_1 \in \{4, 8\}$ and maximum channel depth throughout the architecture $c_{\max} \in \{32, 64, 128\}$, resulting in 966 total architectures. As mentioned before, to obtain the true fitness of each architecture, we train them until convergence three times for different batch sizes (32, 64, 128) and set the fitness to the best validation loss reached during training. We report the mean Spearman rank correlation of ZC proxy score and best validation loss (fitness) over all combinations in Table 3.

Overall, naswot performs the best out of all proxies. Snip and scaled jacob_cov perform approximately as well as naswot, while grasp, Fisher, and synflow are outperformed.

Table 3. Mean Spearman rank correlation of the best validation loss of each architecture and each ZC proxy score z and its scaled variants by layers z_l and channels z_c.

	Grad_norm	**Snip**	**Grasp**	**Fisher**	**Synflow**	**Jacob_cov**	**Naswot**
z	−0.655	−0.684	−0.484	−0.398	−0.252	−0.483	**−0.693**
z_l	0.015	−0.243	−0.064	−0.068	−0.052	−0.729	**0.737**
z_c	0.400	0.201	0.148	0.098	0.047	−0.732	**0.737**

To sum up, no ZC proxy is perfectly robust out of the box for our setting with regards to weight initialization, mini-batch sampling, mini-batch size, and architecture size. After scaling by the number of layers or channels in the architecture, naswot is robust and jacob_cov slightly worse. All other ZC proxies are not robust and therefore unusable for traffic prediction. Scaled naswot and scaled jacob_cov are the most correlated with validation loss. Combining the robustness and correlation results, naswot comes out as the best ZC proxy to use for our search space and task. The robustness with respect to architecture size makes it possible to run scaled naswot on very small versions of the architectures, further lowering computation cost.

2.7. Low Cost Evolutionary Neural Architecture Search

In this Section we incorporate the naswot ZC proxy into the genetic algorithm described in Section 2.4 and evaluate our method on four real world datasets. In the following we describe the four datasets, the evaluation metrics and baseline models we use for comparison.

2.7.1. Datasets

Our experiments are conducted on four real world datasets, two of which are concerned with traffic flow prediction and two with traffic speed prediction:

- PeMSD4–The PeMSD4 dataset is made up of traffic flow measurements from 307 loop detectors in the San Francisco Bay Area within the period from 1 January 2018 to 28 February 2018 [10].
- PeMSD8–The PeMSD8 dataset contains traffic flow measurements from 170 loop detectors in the San Bernardino Area from 1 July 2016 to 31 August 2016 [10].

- METR-LA–The METR-LA dataset includes traffic speed readings at 207 sensors located on the highways of Los Angeles County from 1 March 2012 to 30 June 2012 [31].
- PEMS-BAY–The PEMS-BAY dataset comprises traffic speed data from 325 measurement sites in the Bay Area of California from 1 January 2017 to 31 May 2017 [31].

All datasets are aggregated into 5 min windows, resulting in 288 timestamps per day. For training, the data are normalized by standard normalization for each node and feature. Given a timestamp t, we want to predict the next hour of traffic conditions, i.e., 12 timesteps. The input $\mathbf{X}_t \in \mathbb{R}^{N \times F \times 12}$ to our network is made up of a recent, daily, and weekly segment from the historical data. These segments are defined as follows:

$$\begin{aligned} \mathbf{X}_t^{\text{recent}} &= [X_{t-11}, ..., X_t] \\ \mathbf{X}_t^{\text{daily}} &= [X_{t+1-288}, ..., X_{t+12-288}] \\ \mathbf{X}_t^{\text{weekly}} &= [X_{t+1-7\times288}, ..., X_{t+12-7\times288}] \end{aligned}$$

As can be seen, the recent segment comprises the last hour of data, the daily segment includes data from the same hour to be predicted, but on the day before and the weekly segment contains the same hour we predict, but one week earlier. Additionally, we include data about time of the day and day of the week for the prediction segment. The segments and time information are stacked in the feature dimension of the input, i.e., $\mathbf{X}_t \in \mathbb{R}^{N \times 5 \times 12}$. Stacking in the feature dimension has not been done in previous works. In [8,32] multiple modules with the same architectures and a fusion layer are used, while in [10,15] only the recent segment is used. We have conducted experiments comparing different input methods and concluded that stacking multiple segments in the feature dimension works best for our approach. However, future research might be conducted to set a standard for the task of traffic prediction.

We seperate the datasets into training, validation and test sets with a 7-1-2 ratio. The adjacency matrices are constructed as in previous works by road network distance and Gaussian kernel threshholding [7].

We remark that, due to the choice of inputs, the resulting datasets include fewer samples than in other works, where only the last 12 timesteps are used as inputs [7,10,15]. Therefore, direct comparisons with their results are to be taken with caution. For fair comparison in this work, we evaluate the baseline models on our datasets.

2.7.2. Metrics

We use mean absolute error (MAE), rooted mean squared error (RMSE) and mean absolute percentage error (MAPE) to evaluate our framework and the baselines:

$$\begin{aligned} MAE &= \frac{1}{N}\sum_{i=1}^{N} \mid \hat{y}_i - y_i \mid, \\ RMSE &= \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}, \\ MAPE &= 100\% \times \frac{1}{N}\sum_{i=1}^{N} \frac{\mid \hat{y}_i - y_i \mid}{y_i} \end{aligned}$$

Here, $N, \hat{y}_i$ and y_i respectively refer to the number of samples, predicted values and ground truth values. Since y_i can be zero-valued for some measurements, we only compute MAPE when ground truth is larger than one.

2.7.3. Baselines

We compare our framework against the following models:

- Historical average (HA)–Traffic is modeled as a seasonal process. We predict future timesteps by taking the average over the last n_d (to be determined) days of the same time.
- AGCRN–Adaptive graph convolutional recurrent network captures spatio and temporal dependencies automatically from the data without the need of predefined adjacency matrices for the graph convolution [10].
- Graph WaveNet–Deploys WaveNet [33] and graph convolutions for modelling spatio-temporal graph signals. The adjacency matrix is self-adapting by discovering structures in the data without prior knowledge [7].
- AutoSTG–Gradient-based NAS framework for spatio-temporal prediction. Pan et al. [15] use special modules for capturing spatio-temporal dependencies from meta data of the attributed graph.

We evaluate all baselines on our own dataset as described in Section 2.7.1. To this end, we adapt the publicly available code and conduct the recommended hyperparameter search of each model.

2.7.4. Framework Settings

We apply GA on each of the four datasets until convergence of the algorithm. We have not performed extensive hyperparameter tuning, as we want to show that the algorithm can be applied by non-experts. We use a warmstart size of 1000 genomes to explore a large chunk of the search space in the beginning. This should lead to a high diversity in the population. Afterwards, we decrease the population size to 100 for a faster search time. Note that this is a big increase in population size to previous work [11]. The crossover probability is fixed to 0.9 while mutation probability p_m is adaptively set for each genome based on their rank in the interval [0.05, 0.15]:

$$p_m(A) = 0.15 - \frac{n_p - rank(A)}{n_p} \times 0.1$$

We use the naswot score to rank architectures. As shown in Section 2.6, the naswot score is stable with regards to batch size and architecture size. Hence, we can select them to be small, which will lead to faster search time. Therefore, each network is scored by naswot with a minibatch size of 32, maximum channel size of 32 and starting channel size of 4. We conduct each experiment on one Nvidia GeForce GTX 3090 GPU. The best model architecture is trained until convergence for different starting learning rates (0.02, 0.01, 0.005) and batch sizes (32, 64, 128). Finally, the best performing model on the validation set is used for measuring performance on the test set.

3. Results

The results of the prediction performance for the four datasets are presented in Table 4, where the MAE, RMSE, and MAPE for the 15 min, 30 min, and 60 min horizons are reported. The experiments were conducted twice with different random seeds, except for the LENAS experiments which were conducted thrice.

As anticipated, the simplest model, HA, showed the worst performance on all four datasets due to its inability to capture the complexity of the spatio-temporal data. This model can only capture the general trend of the data and fails to adapt to local changes in the trend.

On the other hand, the two hand-crafted deep learning models, AGCRN and GWN, which can model spatio-temporal dependencies, achieved better predictive performance. However, AGCRN had the worst performance among all deep learning models. GWN outperformed all other methods on the METR-LA dataset and achieved the best or comparable performance to AutoSTG and ENAS while slightly outperforming LENAS on the PEMS-BAY dataset.

We were unable to conduct experiments with AutoSTG on PeMSD4 and PeMSD8 due to the lack of metadata for these datasets. Nonetheless, AutoSTG was the best-performing method on the PEMS-BAY dataset for the 15 min horizon and exhibited comparable performance to other approaches but lacked stability with different random seeds.

The ENAS framework described in [23] outperformed all other approaches on PeMSD4 for all metrics and on most metrics on PeMSD8. However, its performance on METR-LA was underwhelming compared to GWN. For PEMS-BAY, the ENAS framework was able to keep up with GWN and AutoSTG.

As expected, LENAS was unable to outperform ENAS as it searches the architecture space less accurately. The results on METR-LA and PEMS-BAY were competitive with other deep learning models, while the performance was lacking on PeMSD8 and PeMSD4, especially for the shorter horizons.

Regarding search time, ENAS exhibited the worst performance, requiring approximately 300 GPU hours for the smallest dataset (PeMSD8) and approximately 1200 GPU hours for the largest dataset (PEMS-BAY). AutoSTG had to be run multiple times for different combinations of architecture-related hyperparameters, with each run taking around 10 GPU hours for search and 5 h for training the discovered architectures. AGCRN and GWN took the least time since they did not include an architecture search process. LENAS improved ENAS search time to 1–4 GPU hours depending on the dataset with a much larger population size.

Table 4. Traffic forecast performance on PeMSD4, PeMSD8, METR-LA and PEMS-BAY datasets. Here, GWN, ENAS and LENAS respectively denote Graph WaveNet, GA without ZC proxies and our framework.

	MAE			**RMSE**			**MAPE**		
PeMSD4	15 min	30 min	60 min	15 min	30 min	60 min	15 min	30 min	60 min
HA	34.33 ± 0.00	34.33 ± 0.00	34.33 ± 0.00	53.27 ± 0.00	53.27 ± 0.00	53.27 ± 0.00	24.22% ± 0.00%	24.22% ± 0.00%	24.22% ± 0.00%
AGCRN	19.02 ± 0.07	19.88 ± 0.11	21.05 ± 0.36	30.99 ± 0.49	32.66 ± 0.53	34.56 ± 0.12	12.49% ± 0.33%	12.92% ± 0.32%	13.92% ± 0.25%
GWN	18.28 ± 0.04	19.24 ± 0.06	20.95 ± 0.04	29.44 ± 0.09	31.09 ± 0.16	33.83 ± 0.21	11.98% ± 0.03%	12.60% ± 0.02%	13.71% ± 0.00%
ENAS	**17.95** ± **0.01**	**18.77** ± **0.00**	**20.44** ± **0.02**	**29.22** ± **0.01**	**30.67** ± **0.00**	**33.21** ± **0.03**	**11.55**% ± **0.01**%	**12.04**% ± **0.03**%	**13.22**% ± **0.03**%
LENAS	18.42 ± 0.04	19.34 ± 0.06	20.81 ± 0.10	29.62 ± 0.05	31.14 ± 0.07	33.42 ± 0.11	11.82% ± 0.04%	12.43% ± 0.05%	13.43% ± 0.08%
PeMSD8	15 min	30 min	60 min	15 min	30 min	60 min	15 min	30 min	60 min
HA	31.33 ± 0.00	31.33 ± 0.00	31.33 ± 0.00	48.72 ± 0.00	48.72 ± 0.00	48.72 ± 0.00	23.50% ± 0.00%	23.50% ± 0.00%	23.50% ± 0.00%
AGCRN	13.17 ± 0.08	13.67 ± 0.13	14.88 ± 0.23	22.34 ± 0.12	23.66 ± 0.15	25.67 ± 0.19	8.46% ± 0.08%	8.81% ± 0.11%	9.73% ± 0.24%
GWN	13.69 ± 0.15	14.18 ± 0.08	14.98 ± 0.08	21.90 ± 0.13	**23.18** ± **0.05**	**25.03** ± **0.05**	8.81% ± 0.15%	9.17% ± 0.09%	9.94% ± 0.02%
ENAS	**13.14** ± **0.01**	**13.62** ± **0.17**	**14.85** ± **0.18**	**21.77** ± **0.07**	23.18 ± 0.25	25.27 ± 0.15	**8.33**% ± **0.07**%	**8.68**% ± **0.14**%	**9.64**% ± **0.07**%
LENAS	14.19 ± 0.01	14.89 ± 0.07	15.85 ± 0.04	22.48 ± 0.01	23.97 ± 0.10	25.65 ± 0.02	8.88% ± 0.03%	9.35% ± 0.05%	10.21% ± 0.01%
METR-LA	15 min	30 min	60 min	15 min	30 min	60 min	15 min	30 min	60 min
HA	13.66 ± 0.00	13.66 ± 0.00	13.66 ± 0.00	21.28 ± 0.00	21.28 ± 0.00	21.28 ± 0.00	19.82% ± 0.00%	19.82% ± 0.00%	19.82% ± 0.00%
AGCRN	3.38 ± 0.00	4.07 ± 0.00	5.04 ± 0.00	7.48 ± 0.00	9.28 ± 0.00	11.34 ± 0.00	8.46% ± 0.00%	10.39% ± 0.00%	12.90% ± 0.00%
GWN	**2.84** ± **0.01**	**3.22** ± **0.01**	**3.62** ± **0.04**	**5.45** ± **0.03**	**6.44** ± **0.00**	**7.39** ± **0.05**	**7.40**% ± **0.05**%	**8.67**% ± **0.14**%	**10.14**% ± **0.20**%
AutoSTG	3.05 ± 0.24	3.69 ± 0.41	4.60 ± 0.77	5.73 ± 0.29	7.21 ± 0.53	9.02 ± 1.06	7.67% ± 0.48%	9.56% ± 0.86%	11.93% ± 1.43%
ENAS	2.97 ± 0.00	3.40 ± 0.01	3.88 ± 0.01	5.75 ± 0.02	6.78 ± 0.01	7.80 ± 0.01	7.90% ± 0.04%	9.50% ± 0.07%	11.27% ± 0.07%
LENAS	3.06 ± 0.06	3.54 ± 0.06	4.00 ± 0.04	5.94 ± 0.12	7.07 ± 0.20	8.15 ± 0.17	8.14% ± 0.27%	9.85% ± 0.23%	11.45% ± 0.13%
PEMS-BAY	15 min	30 min	60 min	15 min	30 min	60 min	15 min	30 min	60 min
HA	3.28 ± 0.00	3.28 ± 0.00	3.28 ± 0.00	6.54 ± 0.00	6.54 ± 0.00	6.54 ± 0.00	7.99% ± 0.00%	7.99% ± 0.00%	7.99% ± 0.00%
AGCRN	1.41 ± 0.03	1.72 ± 0.01	1.99 ± 0.01	2.95 ± 0.02	3.89 ± 0.01	4.56 ± 0.02	3.09% ± 0.01%	3.99% ± 0.00%	4.79% ± 0.00%
GWN	1.33 ± 0.01	**1.62** ± **0.02**	**1.90** ± **0.02**	2.81 ± 0.02	3.71 ± 0.04	**4.44** ± **0.11**	2.84% ± 0.01%	3.74% ± 0.04%	4.59% ± 0.12%
AutoSTG	1.32 ± 0.05	1.63 ± 0.07	1.93 ± 0.09	2.80 ± 0.10	3.78 ± 0.15	4.59 ± 0.21	2.82% ± 0.15%	3.80% ± 0.26%	4.71% ± 0.32%
ENAS	1.32 ± 0.00	1.63 ± 0.00	1.91 ± 0.00	2.81 ± 0.00	**3.71** ± **0.01**	4.45 ± 0.01	2.82% ± 0.00%	3.74% ± 0.01%	4.59% ± 0.00%
LENAS	1.36 ± 0.00	1.68 ± 0.00	1.95 ± 0.01	2.87 ± 0.01	3.81 ± 0.02	4.53 ± 0.03	2.93% ± 0.01%	3.89% ± 0.01%	4.67% ± 0.01%

4. Discussion and Conclusions

In this research, we explored zero-cost proxies, namely the naswot proxy, to estimate network performance for traffic forecasting tasks. Our novel approach centered on utilizing a performance estimation process, rather than training until convergence, as typically employed in other frameworks such as ENAS.

We observed that our LENAS framework, despite its advantage of fast search times and lower computational costs, displayed worse performance compared to other deep learning models. This underperformance is largely attributed to the disconnect between the naswot score and the validation loss of the architectures. Our results indicated an average Spearman rank correlation of 0.737, as highlighted in Table 3, signifying a substantial margin of error when ranking architectures using the naswot score as opposed to the validation loss.

The lack of correlation between the naswot score and the validation loss was a primary factor contributing to the inaccuracies in our model. Hence, while the naswot proxy, once normalized by network size, proved to be stable concerning weight initialization, mini-batch sampling, and size, its use revealed notable challenges in achieving comparable accuracy with the baseline models that do not incorporate performance estimation.

The experimental trials conducted with two traffic speed benchmarks and two traffic flow benchmarks affirmed the double-edged nature of using the naswot score. On one hand, we managed to speed up the neural architecture search process by two orders of magnitude and explore the architecture space more thoroughly. Conversely, this came at the cost of a decrease in accuracy, which emphasizes the need to balance speed with precision in the application of such zero-cost proxies. When analyzing the experimental results in Table 4, LENAS often performs worse than GWN, a handcrafted approach, suggesting that the inclusion of naswot as performance estimator might be less effective than using GWN. Both methods use adjacency matrices for the graph convolution. The choice of the adjacency matrix can be crucial for the performance of the neural network. LENAS employs a predefined adjacency matrix, while GWN uses an adaptive matrix which adapts to the data at hand. This might be beneficial for the GWN approach and, hence, in future research, LENAS should be extended to include adaptive adjacency matrices or attention mechanisms [34].

In light of our findings, future research endeavors should prioritize designing zero-cost proxies specifically geared towards regression tasks to yield more accurate results. While our LENAS framework showed potential in terms of reduced search time and low computational requirements, the accuracy of the naswot proxy needs further enhancement.

Overall, the exploration of zero-cost proxies such as the naswot score has shown the potential and challenges of such an approach. This work opens up new pathways for evolutionary neural architecture search processes, especially in the field of traffic forecasting, provided the inaccuracies are effectively addressed in future iterations.

Author Contributions: Conceptualization, D.K. and C.B.; methodology, D.K.; software, D.K.; validation, D.K.; formal analysis, D.K.; investigation, D.K.; resources, D.K.; data curation, D.K.; writing—original draft preparation, D.K.; writing—review and editing, D.K.; visualization, D.K.; supervision, C.B.; project administration, D.K.; funding acquisition, C.B. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the European Regional Development Fund (ERDF).

Data Availability Statement: We only used publicly available data, see Section 2.7.1.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Sun, H.; Liu, H.X.; Xiao, H.; He, R.R.; Ran, B. Use of Local Linear Regression Model for Short-Term Traffic Forecasting. *Transp. Res. Rec.* **2003**, *1836*, 143–150. [CrossRef]
2. Makridakis, S.; Hibon, M. ARMA Models and the Box–Jenkins Methodology. *J. Forecast.* **1997**, *16*, 147–163. [CrossRef]

3. Zivot, E.; Wang, J. Vector Autoregressive Models for Multivariate Time Series. In *Modeling Financial Time Series with S-Plus*; Springer: New York, NY, USA, 2003.
4. Mallek, A.; Klosa, D.; Büskens, C. Impact of Data Loss on Multi-Step Forecast of Traffic Flow in Urban Roads Using K-Nearest Neighbors. *Sustainability* **2022**, *14*, 11232. [CrossRef]
5. Mallek, A.; Klosa, D.; Büskens, C. Enhanced K-Nearest Neighbor Model For Multi-steps Traffic Flow Forecast in Urban Roads. In Proceedings of the 2022 IEEE International Smart Cities Conference (ISC2), Pafos, Cyprus, 26–29 September 2022; pp. 1–5. [CrossRef]
6. Fu, R.; Zhang, Z.; Li, L. Using LSTM and GRU neural network methods for traffic flow prediction. In Proceedings of the 2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC), Wuhan, China, 11–13 November 2016; pp. 324–328. [CrossRef]
7. Wu, Z.; Pan, S.; Long, G.; Jiang, J.; Zhang, C. Graph WaveNet for Deep Spatial-Temporal Graph Modeling. In Proceedings of the IJCAI, Macao, 10–16 August 2019.
8. Ge, L.; Li, S.; Wang, Y.; Chang, F.; Wu, K. Global Spatial-Temporal Graph Convolutional Network for Urban Traffic Speed Prediction. *Appl. Sci.* **2020**, *10*, 1509. [CrossRef]
9. Klosa, D.; Mallek, A.; Büskens, C. Short-Term Traffic Flow Forecast Using Regression Analysis and Graph Convolutional Neural Networks. In Proceedings of the 2021 IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys), Haikou, China, 20–22 December 2021; pp. 1413–1418. [CrossRef]
10. Bai, L.; Yao, L.; Li, C.; Wang, X.; Wang, C. Adaptive Graph Convolutional Recurrent Network for Traffic Forecasting. In Proceedings of the NIPS'20': 34th International Conference on Neural Information Processing Systems, Vancouver, BC, Canada, 6–12 December 2020.
11. Pham, H.; Guan, M.; Zoph, B.; Le, Q.; Dean, J. Efficient Neural Architecture Search via Parameters Sharing. In Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; Machine Learning Research; Dy, J., Krause, A., Eds.; Volume 80, pp. 4095–4104.
12. Liu, H.; Simonyan, K.; Yang, Y. DARTS: Differentiable Architecture Search. In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
13. Gao, Y.; Yang, H.; Zhang, P.; Zhou, C.; Hu, Y. Graph Neural Architecture Search. In Proceedings of the IJCAI'20: Twenty-Ninth International Joint Conference on Artificial Intelligence, Online, 7–15 January 2021.
14. Zhou, K.; Song, Q.; Huang, X.; Hu, X. Auto-GNN: Neural Architecture Search of Graph Neural Networks. *arXiv* **2019**, arXiv:1909.03184.
15. Pan, Z.; Ke, S.; Yang, X.; Liang, Y.; Yu, Y.; Zhang, J.; Zheng, Y. AutoSTG: Neural Architecture Search for Predictions of Spatio-Temporal Graph. In Proceedings of the WWW '21: Web Conference 2021, New York, NY, USA, 19–23 April 2021; pp. 1846–1855. [CrossRef]
16. Zoph, B.; Le, Q. Neural Architecture Search with Reinforcement Learning. In Proceedings of the International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
17. Elsken, T.; Metzen, J.H.; Hutter, F. Efficient Multi-Objective Neural Architecture Search via Lamarckian Evolution. In Proceedings of the 7th International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
18. Lopes, V.; Alirezazadeh, S.; Alexandre, L.A. EPE-NAS: Efficient Performance Estimation Without Training for Neural Architecture Search. In *Artificial Neural Networks and Machine Learning–ICANN 2021*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 552–563. [CrossRef]
19. White, C.; Zela, A.; Ru, B.; Liu, Y.; Hutter, F. How Powerful are Performance Predictors in Neural Architecture Search? *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 28454–28469. [CrossRef]
20. Vlahogianni, E.I.; Karlaftis, M.G.; Golias, J.C. Optimized and meta-optimized neural networks for short-term traffic flow prediction: A genetic approach. *Transp. Res. Part C Emerg. Technol.* **2005**, *13*, 211–234. .: 10.1016/j.trc.2005.04.007. [CrossRef]
21. Rahimipour, S.; Moienfar, R.; Hashemi, S.M. Traffic Prediction Using a Self-Adjusted Evolutionary Neural Network. *J. Mod. Transp.* **2018**, *27*, 306–316. [CrossRef]
22. Li, L.; Qin, L.; Qu, X.; Zhang, J.; Wang, Y.; Ran, B. Day-ahead traffic flow forecasting based on a deep belief network optimized by the multi-objective particle swarm algorithm. *Knowl.-Based Syst.* **2019**, *172*, 1–14. [CrossRef]
23. Klosa, D.; Büskens, C. Evolutionary Neural Architecture Search for Traffic Forecasting. In Proceedings of the 21st IEEE International Conference on Machine Learning and Applications, to Appear in IEEE Xplore, Nassau, Bahamas, 12–15 December 2022; pp. 1230–1237.
24. Yu, F.; Koltun, V. Multi-Scale Context Aggregation by Dilated Convolutions. In Proceedings of the International Conference on Learning Representations (ICLR), San Juan, Puerto Rico, USA, 2–4 May 2016.
25. Abdelfattah, M.S.; Mehrotra, A.; Dudziak, L.; Lane, N.D. Zero-Cost Proxies for Lightweight NAS. *arXiv* **2021**. [CrossRef]
26. Lee, N.; Ajanthan, T.; Torr, P.H.S. SNIP: Single-shot Network Pruning based on Connection Sensitivity *arXiv* **2018**. [CrossRef]
27. Wang, C.; Zhang, G.; Grosse, R. Picking Winning Tickets Before Training by Preserving Gradient Flow. *arXiv* **2020**. [CrossRef]
28. Tanaka, H.; Kunin, D.; Yamins, D.L.K.; Ganguli, S. Pruning neural networks without any data by iteratively conserving synaptic flow. *arXiv* **2020**. [CrossRef]

29. Theis, L.; Korshunova, I.; Tejani, A.; Huszár, F. Faster gaze prediction with dense networks and Fisher pruning. *arXiv* **2018**. [CrossRef]
30. Mellor, J.; Turner, J.; Storkey, A.; Crowley, E.J. Neural Architecture Search without Training. *arXiv* **2020**. [CrossRef]
31. Li, Y.; Yu, R.; Shahabi, C.; Liu, Y. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
32. Guo, S.; Lin, Y.; Feng, N.; Song, C.; Wan, H. Attention Based Spatial-Temporal Graph Convolutional Networks for Traffic Flow Forecasting. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 922–929. [CrossRef]
33. van den Oord, A.; Dieleman, S.; Zen, H.; Simonyan, K.; Vinyals, O.; Graves, A.; Kalchbrenner, N.; Senior, A.; Kavukcuoglu, K. WaveNet: A Generative Model for Raw Audio. In Proceedings of the 9th ISCA Workshop on Speech Synthesis Workshop (SSW 9), Sunnyvale, CA, USA, 13–15 September 2016; p. 125.
34. Cai, L.; Janowicz, K.; Mai, G.; Yan, B.; Zhu, R. Traffic transformer: Capturing the continuity and periodicity of time series for traffic forecasting. *Trans. GIS* **2020**, *24*, 736–755. [CrossRef]