DEPARTMENT OF
TRANSPORTATION SYSTEMS
ENGINEERING

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Transport and Logistics

An Image Fusion Framework for Deep Learning in Traffic Forecasting

Xiahan LI

DEPARTMENT OF
TRANSPORTATION SYSTEMS
ENGINEERING

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Transport and Logistics

An Image Fusion Framework for Deep Learning in Traffic Forecasting

Ein Image Fusion Framework für Deep Learning in der Verkehrsprognose

Author: Xiahan LI
Supervisor: Dr. Tao Ma (tao.ma@tum.de)
Advisor: Prof. Dr. Constantinos Antoniou (c.antoniou@tum.de)
Submission Date: May. 20th, 2021

I confirm that this master's thesis in transport and logistics is my own work and I have documented all sources and material used.

Munich, May. 20th, 2021 Xiahan LI

Acknowledgments

I would like first to express my gratitude to Prof. Dr. Constantinos Antoniou, the Chair of Transportation Systems Engineering leader at the Technical University of Munich (TUM), for his acceptance to conduct this master thesis research at the Chair.

Most notably, I wish to express my sincere appreciation to my supervisor, Dr. Tao Ma, from the Chair of Transportation Systems Engineering. He selflessly offered his advice, time, vast knowledge, and experience to fine-tuning my topic. Although some supervisors are willing to provide me with some available thesis topics, I still decide to propose a thesis topic by myself. Because I want to improve my scientific research level by independently raise questions and innovatively solve them. When I talked with Dr. Tao Ma about my proposal, Dr. Tao Ma praised me for my innovation and strongly supported my own topic.

I would like to express my sincere gratitude to DiDi Chuxing GAIA Open Dataset Initiative in China, for sharing the data and giving research students like me the opportunity to use their datasets for academic purposes. The request to use the "Chengdu City Second Ring Road Regional Trajectory Dataset" and “Travel Time Index Dataset” for this thesis was quickly authorized and accepted by them. I also want to express my sincere gratitude to my bachelor’s University, Shanghai Jiao Tong University, to allow me to use the computation facilities in the high-performance computing platform of SJTU Student Innovation Center to complete my master thesis. Sincerely appreciation will also be given to Kaggle, one of the world’s largest data science community with powerful tools and resources to help me not only finish my master thesis recently but also acquire ample data science knowledge and achieved Competition Expert two years ago.

During my two years of master’s study of Transport and Logistics in TUM-Asia, I would like to thank all TUM-Asia and TUM teaching faculties and staff, like Dr. Fritz Busch, Dr. Peter Klaus, Dr. Bernhard Lechner, Dr. Andreas Rau, Mr. Xiaodong Liu, and so on. Because of their guidance and support. Also, thanks for the two scholarships that TUM awarded to me.

Here I have a special thanks for my previous department boss and company supervisor, Dr. Xu. Chen and Dr. Bo. Sui, in Winning Health Co. Ltd, Shanghai,

China. At that time I was a new-grad from my bachelor with insufficient practical working skill as an Artificial Intelligence Engineer, but both of them help me and guide me a lot in not only the technical working skill but also the interpersonal communication skill.

Conclusively, I would like to express my deepest gratitude and special regards to my parents, Mr. Canhong Li and Mrs. Xiaochun Xia, for their advice, encouragement, and motivation from the beginning of this master's program to the successful completion of this thesis.

Abstract

The rapid development of Intelligent Transport System promotes the demand for traffic forecasting in much more efficient ways than before. Based on big data, one of the efficient ways of traffic forecasting is using deep learning methods. Although some of the researchers successfully apply deep learning in traffic forecasting, improvements of innovative feature engineering tools are expected in further researches.

In this thesis, the main objective is to implement various models to predict the values of Travel Time Index(TTI), whose time range is 24 hours and the positions are located in several roads and areas in Chengdu City, China. The raw datasets mainly consist of both spatial and temporal features of transportation status in Chengdu City, China, but their inter-datasets relationships are vague. The innovative point of this thesis is that the author establishes an image fusion framework to merge the datasets derived from different sources.

The main processes of the prediction are: First, analyze the basic structures and statistical distributions of three raw datasets: Network Dataset, Routing Dataset , Travel Time Index and Average Speed Dataset. The outliers in the datasets are also analyzed and discussed. Second, under some appropriate assumptions, establish the image fusion framework to fuse the previous three datasets into two sets of images: Roadmap Images and Trajectory Images, and fuse them with Geographical Images into a set of three phases RGB images according to timestamps. Third, do some preparations for the deep learning processes. Fourth, use different models to predict the values of Travel Time Indexes and compare their performances by setting the Historical Average results as the baseline.

The results show that Resnet34 models whose inputs are images generated by the image fusion framework can improve the performance by 9% compared to the baseline. Besides, the robustness of the performance of the Resnet34 model is also good in predicting various roads. Because the image fusion framework can augment the dataset and establish the relationship of spatial features between different datasets.

Kurzfassung

Die rasante Entwicklung des intelligenten Verkehrssystems fördert die Nachfrage nach Verkehrsprognosen wesentlich effizienter als zuvor. Basierend auf Big Data besteht eine der effizienten Methoden zur Verkehrsprognose darin, Deep-Learning-Methoden zu verwenden. Obwohl einige der Forscher erfolgreich Deep Learning in der Verkehrsprognose anwenden, werden in weiteren Untersuchungen Verbesserungen innovativer Tools für das Feature-Engineering erwartet.

In dieser Arbeit besteht das Hauptziel darin, verschiedene Modelle zu implementieren, um die Werte des Reisezeitindex (TTI) vorherzusagen, dessen Zeitbereich 24 Stunden beträgt und dessen Positionen sich auf mehreren Straßen und Gebieten in Chengdu City, China, befinden. Die Rohdatensätze bestehen hauptsächlich aus räumlichen und zeitlichen Merkmalen des Transportstatus in Chengdu City, China, aber ihre Beziehungen zwischen den Datensätzen sind vage. Der innovative Punkt dieser Arbeit ist, dass der Autor ein Bildfusions-Framework erstellt, um die aus verschiedenen Quellen abgeleiteten Datensätze zusammenzuführen.

Die Hauptprozesse der Vorhersage sind: Analysieren Sie zunächst die Grundstrukturen und statistischen Verteilungen von drei Rohdatensätzen: Netzwerkdatensatz, Routingdatensatz, Reisezeitindex und Durchschnittsgeschwindigkeitsdatensatz. Die Ausreißer in den Datensätzen werden ebenfalls analysiert und diskutiert. Zweitens legen Sie unter geeigneten Voraussetzungen das Bildfusions-Framework fest, um die vorherigen drei Datensätze zu zwei Bildsätzen zusammenzuführen: Roadmap-Bilder und Trajektorienbilder, und verschmelzen sie mit geografischen Bildern zu einem Satz von drei Phasen-RGB-Bildern gemäß Zeitstempeln. Drittens machen Sie einige Vorbereitungen für die tiefen Lernprozesse. Viertens: Verwenden Sie verschiedene Modelle, um die Werte von Reisezeitindizes vorherzusagen und ihre Leistung zu vergleichen, indem Sie die Ergebnisse des historischen Durchschnitts als Basis festlegen.

Die Ergebnisse zeigen, dass Resnet34-Modelle, deren Eingaben Bilder sind, die vom Bildfusionsframework generiert wurden, die Leistung im Vergleich zur Basislinie um 9% verbessern können. Außerdem ist die Robustheit der Leistung des Resnet34-Modells auch gut für die Vorhersage verschiedener Straßen. Weil das Bildfusions-Framework den Datensatz erweitern und die Beziehung zwischen räumlichen Merkmalen zwischen verschiedenen Datensätzen herstellen kann.

Contents

1. Introduction

1.1. Background Introduction

1.1.1. Introduction of Intelligent Transport System

Intelligent Transport Systems(ITS) means systems in which information and communication technologies are applied in the field of road transport, including infrastructure, vehicles, users, and in traffic management and mobility management, as well as for interfaces with other modes of transport. In the field of ITS, mainly three kinds of data are concerned by researchers: road data, traffic data, and travel data[1].

•Road data: data on road infrastructure characteristics, including fixed traffic signs or their regulatory safety attributes.

•Traffic data: historic and real-time data on road traffic characteristics.

•Travel data: basic data such as public transport timetables and travel time.

Based on the information provided by the data introduced above, researchers can do traffic forecast by not only extracting and analyzing features from the data but also expanding and augmenting some related data, like geographical data, to gain a new perspective to interpret the data.

1.1.2. Introduction of Traffic Forecast

The main target of traffic forecast is to predict the future flow using previous traffic speeds as considering the other parameters such as road conditions[2].

Traffic forecast is widely used in multi-aspects of social development:

For government:

•Design the capacity of infrastructure, like the width of a bridge to be constructed.

•Estimate the financial and social viability of projects.

•Calculate environmental impacts due to traffic.

For drivers:

•Choose the optimal route for drivers to avoid congestion and accidents.

•Provide estimated traveling time to help drivers to make travel arrangements and decisions.

In terms of traffic forecast models, Van Lint and Hoogendoorn classified short-term traffic forecast methods into three categories: naïve methods, parametric models and non-parametric models[3]:

•Naïve methods: naïve methods is the forecasting methods without any model assumption. Naïve methods are easy to implement and require low computation effort, but the accuracy is usually very low.

List of examples: Instantaneous; Historical Averages; Clustering.

•Parametric models: the structures of parametric models are predetermined, but parameters of the model are needed to be estimated using data. Parametric models can model 'unseen' cases such as traffic incidence. Compared to non-parametric models, parametric models usually need fewer data and sometimes shown good accuracy.

List of examples: Traffic simulation models; Time series.

•Non-parametric models: both the model structure and model parameters are determined from data. Non-parametric models can model the difficult, dynamic, and non-linear processes found in traffic. However, most of the studies focus on prediction on one single location or one route due to the lack of data on all roads.

List of examples: k-Nearest Neighbor; Locally Weighted Regression; Fuzzy Logic; Bayesian networks; Neural networks.

Among all the traffic forecast methods indicated above, neural networks stand out from the crowd due to the rapid development of deep learning over the past decades.

1.1.3. Introduction of Deep Learning and Computer Vision

Deep Learning is part of a broader family of machine learning methods based on artificial neural networks. Deep Learning is used in the domain of digital image processing to solve difficult problems (e.g. image colorization, classification, segmentation, and detection). Deep Learning methods such as Convolutional Neural Networks (CNNs) mostly improve prediction performance using big data and

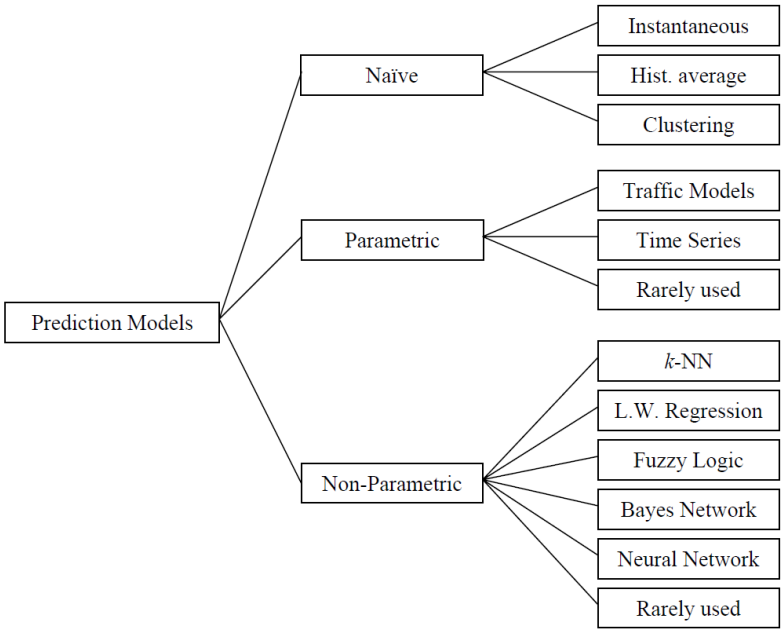


Figure 1.1.: A tree diagram of some prediction models[3].

plentiful computing resources and have pushed the boundaries of what was possible. Problems that were assumed to be unsolvable are now being solved with super-human accuracy[4].

Comparing to traditional computer vision, neural networks used in deep learning are trained rather than programmed, so they require less expert analysis and fine-tuning, they can also exploit the big data better than traditional methods in today's system. Deep learning also provides superior flexibility because CNN models and frameworks can be re-trained using a custom dataset for any use case, while the traditional computer vision algorithms tend to be more domain-specific[4].

1.1.4. Objectives

Five objectives will be achieved, among which objectives 1 to 4 are related to feature engineering, and objective 5 is related to deep learning.

1)Use image processing techniques to transform textual and digital data to graphical data.

2)Build up an image fusion framework to combine different image data.

3)Actively decide uncertain parameters when encountering dilemmas in feature engineering.

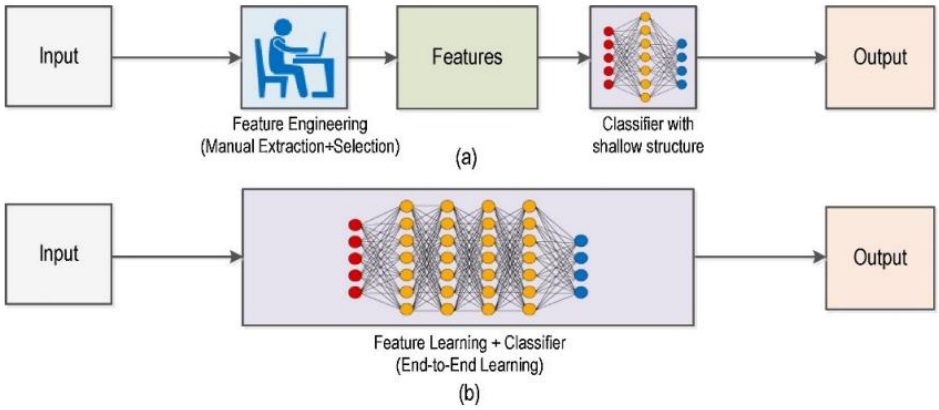


Figure 1.2.: Comparison of (a) traditional computer vision workflow vs (b) deep learning workflow[4].

4)Propose a convenient formula of macroscopic level Travel Time Index from microscopic level Travel Time Index under appropriate assumptions.

5)Use some neural network as examples to train, validate and test the traffic forecast, then comment and discuss the results.

1.1.5. Innovative Point

The second objective above is an innovative point. The author of this thesis establishes a graphical image fusion framework to combine Geographical, Roadmap, and Trajectory Image. The output of the framework is a series of 3 phases traffic images that can be trained in computer vision neural networks to do traffic prediction.

When encountering a textual transportation dataset, most of the researchers will follow the layout of the textual document and use forecasting tools to fit the data. However, by fitting the data into the innovative framework, researchers can mine the features of data and interpret the data from a new perspective: graphical interpretation of textual data.

1.1.6. Motivation

This thesis will emphasize more feature engineering than deep learning. From the perspective of the author, good feature engineering represents a good understanding and interpretation of features in data. Feature engineering is an engineering process, which requires multifaceted capabilities and interdisciplinary knowledge of computer science, mathematics, transportation engineering, mechanical engineering, and so on. Thus, innovative techniques from the multifaceted field like transportation under the

data science and machine learning will be highly recommended in the future

Under the situation stated above, the author proposes this research, trying to interpret the features of data from a new perspective and establish an innovative tool in feature engineering.

1.2. Research Framework

The thesis consists of six main chapters with bibliography and appendix as additional information:

1.2.1. Research Framework of Chapter 1

Chapter 1 consists of Background Introduction, Objectives, Innovation Point, and Motivation.

- In Background Introduction, basic information of Intelligent Transport System, traffic forecast and deep learning in computer vision will be introduced.
- In Objectives, 4 objectives in feature engineering and 1 objective in deep learning are listed.
- In Innovation Point, the innovation point of this research will be specifically illustrated.
- In Motivation, the author of this thesis will state how he come up with this researches and what is his perspective about traffic forecast.

1.2.2. Research Framework of Chapter 2

Chapter 2 consists of a Literature Review of tools and methods from both feature engineering and computer vision deep learning.

In feature engineering, the literature review of computer-aided design(CAD) geographical layout, geographical and time representation, data fusion for multispectral images, and travel time index (TTI) will be demonstrated.

In deep learning, the literature review of convolutional neural network(CNN) and deep residual network(Resnet), recurrent neural network(RNN) and long short term memory(LSTM), K-fold cross-validation and learning rate reduction.

•In computer-aided design(CAD) geographical layout, basic computer-aided design(CAD) concept and geographic layout dataset of metropolis in the form of CAD will be introduced.

•In geographical and time representation, two geographical representations, Gcj-02 coordinates and Well-known text (WKT) representation of geometry will be introduced. One time representation, Unix time, will be introduced.

•In data fusion for multispectral images, the advantages of data fusion for multispectral images are concisely illustrated, and an example of geographical data fusion is shown.

•In Travel Time Index (TTI), an index that is widely used in the evaluation of urban congestion degree called Travel Time Index (TTI) is introduced, and the mathematical representation of Travel Time Index is also stated.

•In a convolutional neural network(CNN) and deep residual neural network (Resnet), the basic structure and working mechanism of basic convolutional neural networks and a deep residual neural network will be introduced.

•In a recurrent neural network(RNN) and long short term memory (LSTM), the basic structure and working mechanism of basic recurrent neural network and long short term memory will be introduced.

•In K-fold Cross-Validation, the process and advantages of a splitting method of training dataset and validation dataset called K-fold cross-validation will be introduced.

•In Learning Rate Reduction, the relationship between the value of learning rate and epochs needed for converging will be shown, which introduces the method of learning rate reduction.

1.2.3. Research Framework of Chapter 3

Chapter 3 consists of Overview and Exploratory Data Analysis of Geographical AutoCAD Data, Network Data, Routing Data, and Travel Time Index. Among the Exploratory Data Analysis, raw data illustration, statistical description of data, and discussion of the data will be applied to each dataset.

•In Overview of All Data, the source, and structure of data mentioned above are briefly illustrated.

•In the part of raw data illustration, the basic descriptions of the dataset, like time range, time interval, and space range of the research area, are listed for each correlated dataset. The descriptions, explanations, and examples of all variables that emerge in each dataset are also listed.

•In the part of the statistical description, we use statistical tools to investigate the data structure. In this part, we not only list the basic statistical indexes like average, variance, and so on for the distribution of data, but also investigate outliers that exist in the data. The graphical illustration like histograms and box plots are also assisting to statistically describe the data.

•In the part of the discussion of data, we comment on the distribution of data with the help of statistical indexes listed previously. We also comment on the outliers and their possible influence on the whole research.

1.2.4. Research Framework of Chapter 4

Chapter 4 is the main chapter in this thesis, which shows the whole process of establishing the image fusion framework and uses multiple models includes deep learning models to validate the performance of the framework. Chapter 4 consists of Assumptions, the process of feature engineering, and deep learning.

•In Assumptions and Pre-settings, several assumptions and pre-settings considered in both actual transport scenes and computation situations are made. They are reduction of precision of coordinates, ignore the width of the road, and restriction of the research area.

Among the process of feature engineering, three layers of inputs: geographical image, roadmap image, trajectory images, and the image fusion process of them, with the output: Travel Time Index, have been generated from the datasets mentioned in Chapter 3. For the three layers of input images and the image fusion framework, we mainly talk about the description of them, generation of them, and demonstration of them. Besides, details about the decision of some parameters are also discussed. For the output Travel Time Indexes, first, we describe how to use the Travel Time Indexes as output values, then we discuss Travel Time Indexes in both individual roads and in a specific area.

•In the description of the 3 kinds of input images, the mapping relationship between the raw data and the images will be described. For example, the relationship between the average speeds in raw data and the values of pixels in images will be described.

•In the description of the image fusion framework, the mapping relationship between the 3 kinds of input images and the RGB 3 phase images will be described.

•In the decision of width of plotting lines and decision of trajectory time duration, two key parameters when plotting the images are decided.

•In the generation of 3 kinds of images and RGB 3 phase images, we list the processes of image generation and use flow charts to vividly depict the processes.

•In a demonstration of 3 kinds of images and RGB 3 phase images, we demonstrate the same kinds of images at different times. We also make a comparison between the images and discuss the difference of their features.

•In TTI for a single road, we list the identification number of target roads, Chinese and English name of the roads, and the color of roads in a demonstration graph.

•In TTI for a specific area, first we deduce the mathematical formula of weight average Travel Time Index for a collection of roads inside a given area. Then we assign the area based on the map. At last, we list the steps of calculating the areal Travel Time Indexes.

Among the process of deep learning, first, we illustrate the basic setting of neural network training, then some preparation processes before deep learning are executed: dataset splitting, loss function choosing, and learning rate reduction in training. After the preparation processes, three models are applied to the TTI value prediction: historical average model, single long short term memory (LSTM), and deep residual neural network (Resnet), among which the only Resnet apply for the RGB 3 phase images generated by the image fusion framework. For the single LSTM, we use two sets of inputs to validate the model.

•In the basic setting of neural network training, we first list the parameters of configurations in two computation workstations used in this research. Then we briefly introduce the deep learning framework in our programming and its advantages.

•In dataset splitting and K-fold cross-validation, we split the dataset into a training dataset, validation dataset, and test dataset, among which training dataset and validation dataset are split under the criteria of K-fold cross validation.

•In loss function choosing, two kinds of widely used loss functions are described and their applicability related to our research are discussed.

•In learning rate reduction in training, a learning rate scheduler will be introduced to reduce the learning rate as the number of epochs increase in training.

•In the historical average baseline, we use the historical average to predict the Travel Time Indexes, and its performance is regarded as a baseline to check the performance of the deep learning models.

•In single LSTM without any data fusion framework, we use the LSTM model to predict the Travel Time Indexes with two different sets of inputs: one is historical Travel Time Indexes, the other is multiple input features related to Travel Time Indexes.

•In Resnet with image fusion framework, we construct a Resnet framework to

predict the Travel Time Indexes with the RGB 3 phase images generated by image fusion framework as the inputs.

1.2.5. Research Framework of Chapter 5

Chapter 5 shows the result of the research. Chapter 5 consists of test loss comparison of different models, graphical demonstration of results, and demonstration of learning procedure of deep learning models.

•In test loss comparison of different models, the results of LSTM with historical Travel Time Indexes values input, multiple values input and use test dataset to predict test dataset are shown.

•In the graphical demonstration of results, we plot the graphs of results generated by models indicated above.

•In the demonstration of the learning procedure of deep learning models, we show the process of model convergence by depicting graphs of results from the Resnet34 model trained by 30 epochs, 60 epochs, and 90 epochs individually.

1.2.6. Research Framework of Chapter 6

Chapter 6 make the conclusion, discussion, and expectation of the research result.

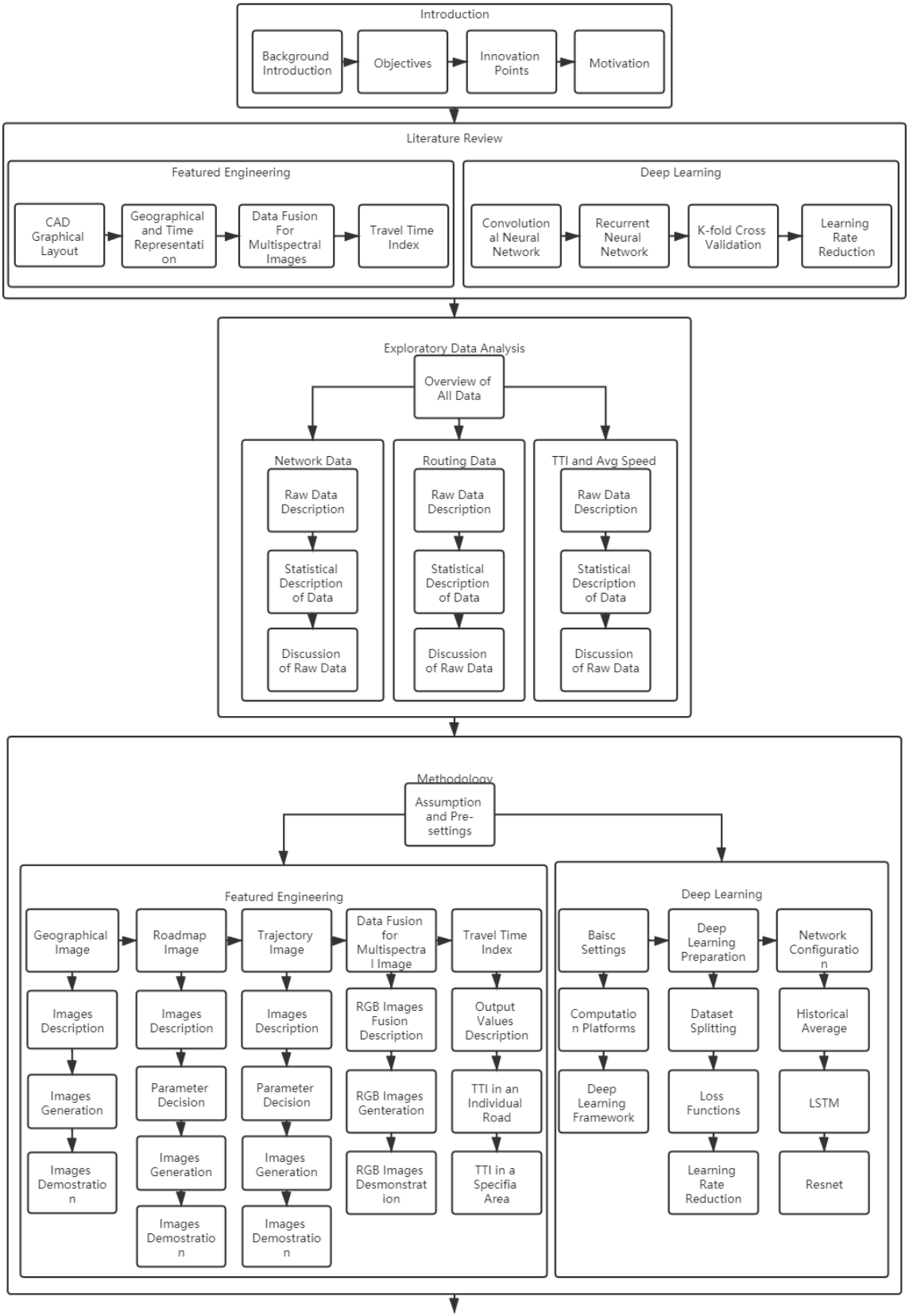
•In conclusion, first we make an overall conclusion of the results in the previous chapter, then we analyze the failure of the single LSTM model. At last, we analyze the advantages and disadvantages of image fusion framework.

•In discussion, first we list four shortcomings of the image fusion frameworks: omit the feature engineering of outliers, low precision of individual roads, unbalanced weights of 3 phases, and bad feature engineering of speed. Then we list the corresponding improvements of the shortcomings: search the possibility of TTI outliers, increase the precision by image segmentation, assign suitable weights of each phase, and improve the feature engineering of speed.

•In expectation, we talk about the expectation of future development of the image fusion framework and the development of the affiliate dataset.

•In personal comment, the author of this thesis made some comments about the research.

1.2.7. Flow Chart of Research Procedure


Introduction
Background Introduction
Objectives
Innovation Points
Motivation
Literature Review
Featured Engineering
CAD Graphical Layout
Geographical and Time Representation
Data Fusion For Multispectral Images
Travel Time Index
Deep Learning
Convolutional Neural Network
Recurrent Neural Network
K-fold Cross Validation
Learning Rate Reduction
Exploratory Data Analysis
Overview of All Data
Network Data
Raw Data Description
Statistical Description of Data
Discussion of Raw Data
Routing Data
Raw Data Description
Statistical Description of Data
Discussion of Raw Data
TTI and Avg Speed
Raw Data Description
Statistical Description of Data
Discussion of Raw Data
Methodology
Assumption and Pre-settings
Featured Engineering
Geographical Image
Images Description
Images Generation
Images Demostration
Roadmap Image
Images Description
Parameter Decision
Images Generation
Images Demostration
Trajectory Image
Images Description
Parameter Decision
Images Generation
Images Demostration
Data Fusion for Multispectral Image
RGB Images Fusion Description
RGB Images Genteration
RGB Images Desmonstration
Travel Time Index
Output Values Description
TTI in an Individual Road
TTI in a Specifia Area
Deep Learning
Baisc Settings
Computation Platforms
Deep Learning Framework
Deep Learning Preparation
Dataset Splitting
Loss Functions
Learning Rate Reduction
Network Configuration
Historical Average
LSTM
Resnet

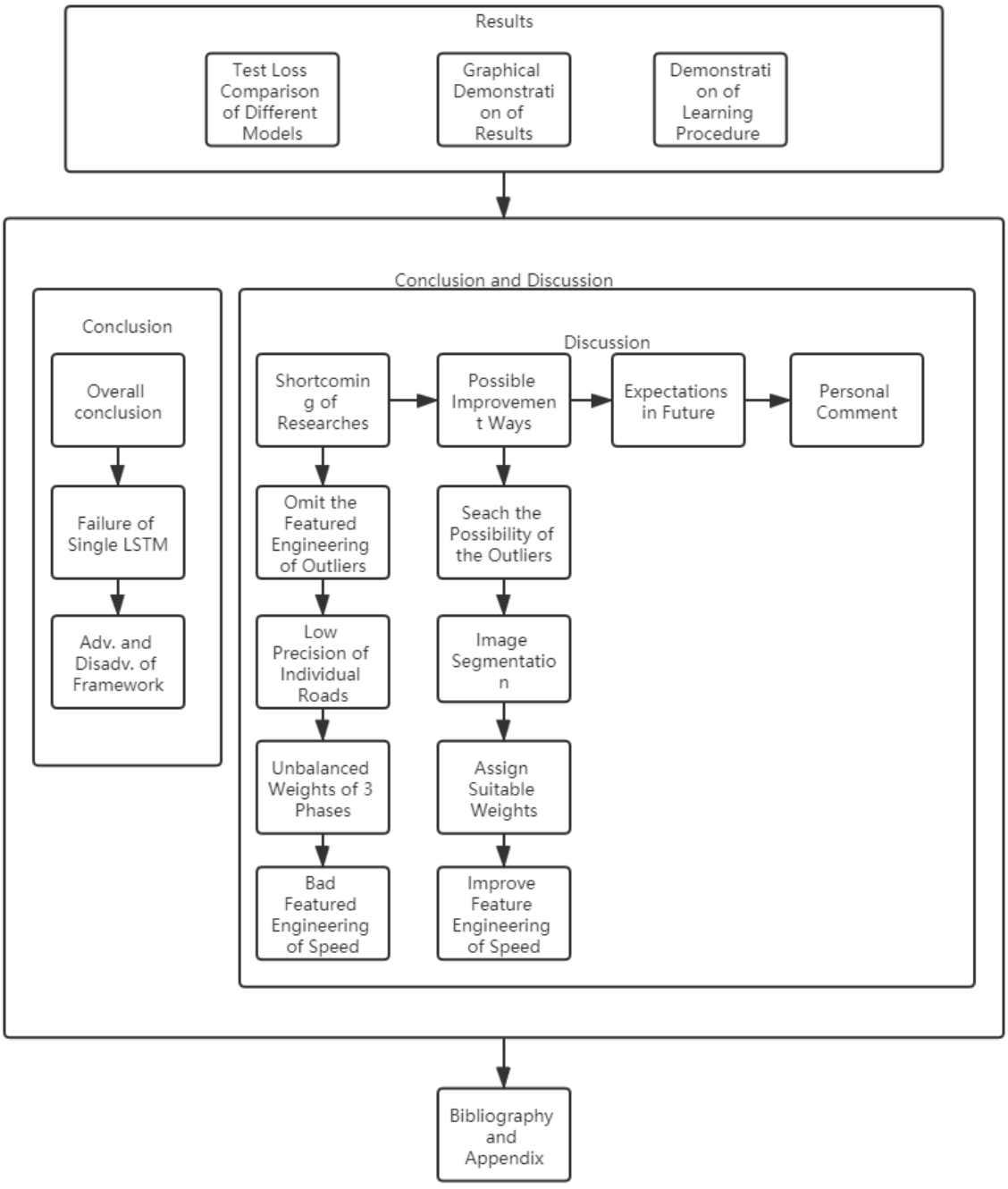


Figure 1.3.: Flow chart of research procedure

2. Literature Review

2.1. Feature Engineering

2.1.1. Introduction of Feature Engineering

Feature Engineering is a data preparation process that can improve the performance of deep learning. By combining and transforming existing features into new features, feature engineering can let the algorithms identify more patterns[5]. In a typical computer vision task, features should be extracted from graphs and represented as capable data in machine learning.

In one of the slides released by Stanford AI Lab, Stanford feature learning methods improves the accuracy of multiple deep learning tasks[6]. Appropriate application of feature engineering will give researchers a new perspective to understand data, thus researchers can figure out potential key features from raw data and improve the accuracy of the result.

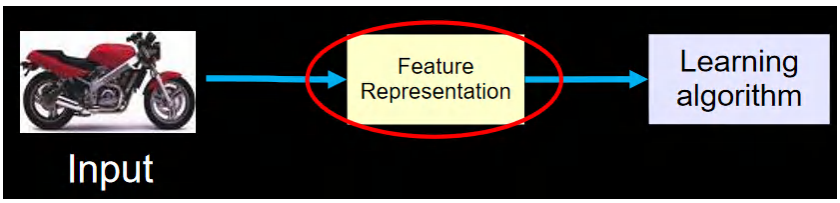


Figure 2.1.: Feature representation in computer vision[6].

2.1.2. Computer-aided design (CAD) geographical layout

CAD, or computer-aided design and drafting (CADD), is a technology for design and technical documentation, which replaces manual drafting with an automated process[7]. CAD is widely used in engineering. In mechanical engineering, engineers use CAD to design and draft the layout of machinery parts. In civil engineering, engineers use CAD to check the geographical layout of cities' layouts.

Cadmapper website, owned or operated by Protomaps LLC, provide free 2D CAD files of over 200 cities' layout in the world. These CAD files are huge and

simplified two-dimensional DXF format files of metropolitan area road networks, which includes urban road, expressway, railway, bodies of water, etc[8].

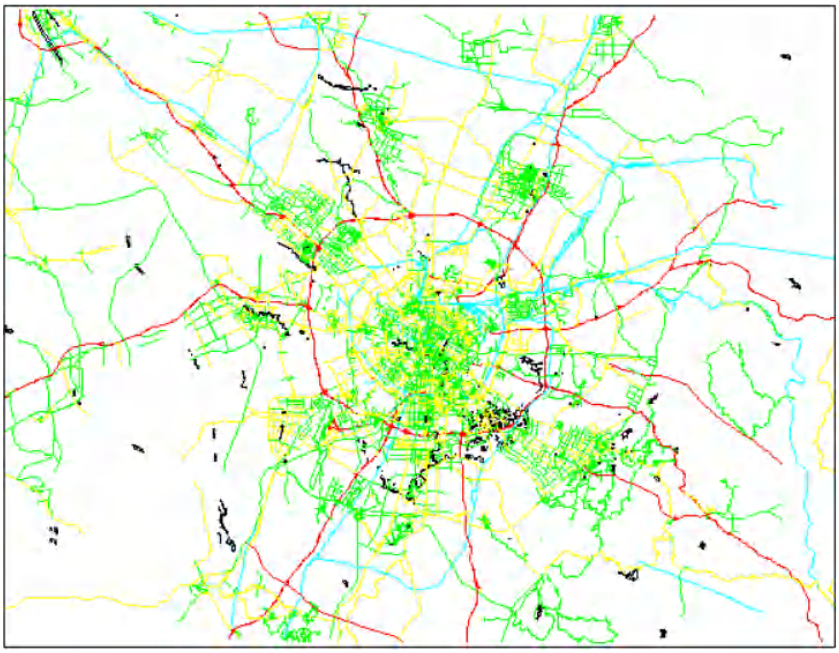


Figure 2.2.: An example of CAD image for chengdu city[8].

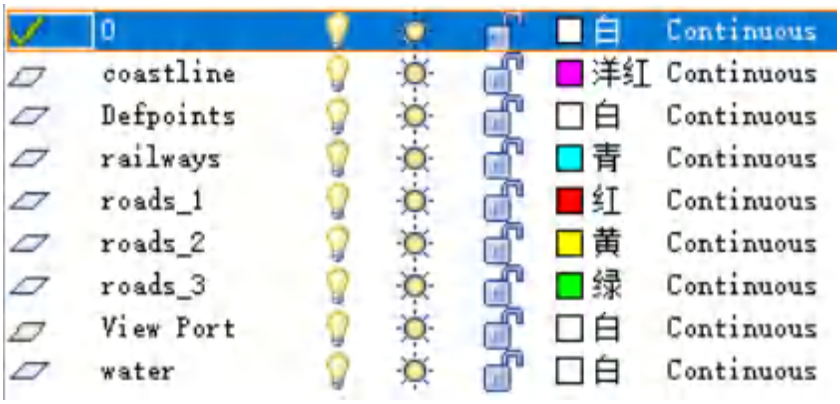


Figure 2.3.: Labels of geographical illustration for CAD image.

2.1.3. Geographical and Time Representation

Gcj-02 Coordinate

GCJ-02, also called Mars Coordinate, is a modified geodetic system based on WGS-84(World Geodetic System), which is widely used in China.

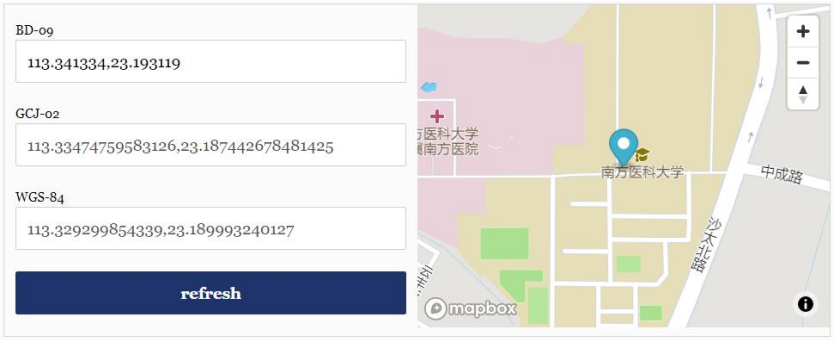


Figure 2.4.: Comparison of longitude and latitude at identical location between three coordinates: BD-09, Gcj-02 and WGS-84[9].

Well-known text(WKT) representation of geometry

Well-known text (WKT) is a text markup language for representing vector geometry objects, which can transfer and store the same information in a more compact form convenient for computer processing but that is not human-readable. WKT representation of geometry can represent distinct geometric objects like points, linestrings, polygons, etc[10].

Geometry Type Text Representations

Geometry Type	Well Known Text (WKT)
POINT	POINT(10 20)
MULTIPOINT	MULTIPOINT(10 20, 15 15, 20 15)
LINESTRING	LINESTRING (10 30, 15 15, 25 40)
MULTILINESTRING	MULTILINESTRING ((40 40, 30 30), (15 15, 9 9))
POLYGON	POLYGON ((10 10, 40 10, 40 30, 50 30, 50 50, 30 50, 30 40, 10 40, 10 10))
MULTIPOLYGON	MULTIPOLYGON (((40 30, 50 30, 50 50, 30 50, 30 40, 40 40, 40 30)), ((10 10, 40 10, 40 30, 30 30, 30 40, 10 40, 10 10)))
COLLECTION	GEOMETRYCOLLECTION(POINT(10,20), LINESTRING(40 40, 30 30), POLYGON((15 15, 30 15, 30 30, 15 30, 15 15)))

Figure 2.5.: Some examples of Well-Known Test geometry[11].

Unix Time

Unix time is a single signed number that increments every second since January 1st, 1970 at 00:00:00 UTC, which makes it easier for computers to store and manipulate than conventional date systems[12].

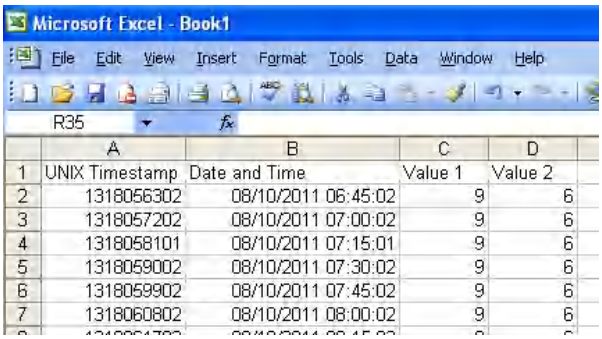


Figure 2.6.: Examples of Converting UNIX Timestamps to Excel Dates[13]

2.1.4. Data Fusion For Multispectral Images

Data fusion is an effective way of synergistically combining information from various sources to better understand a given scene. For example, the fusion of multi-spectral and synthetic aperture radar images could retain the advantages of each data, hence benefiting accurate land cover classification[14]. Comparing to insufficient image data due to the limitation of imaging equipment, multi-spectral data have relatively rich spectral information[14].

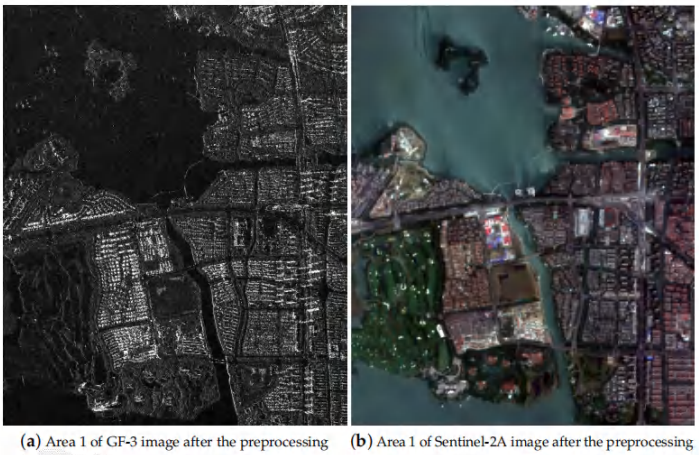

(a) Area 1 of GF-3 image after the preprocessing (b) Area 1 of Sentinel-2A image after the preprocessing

Figure 2.7.: Two Images Depict The same area acquired from two different sources[14].

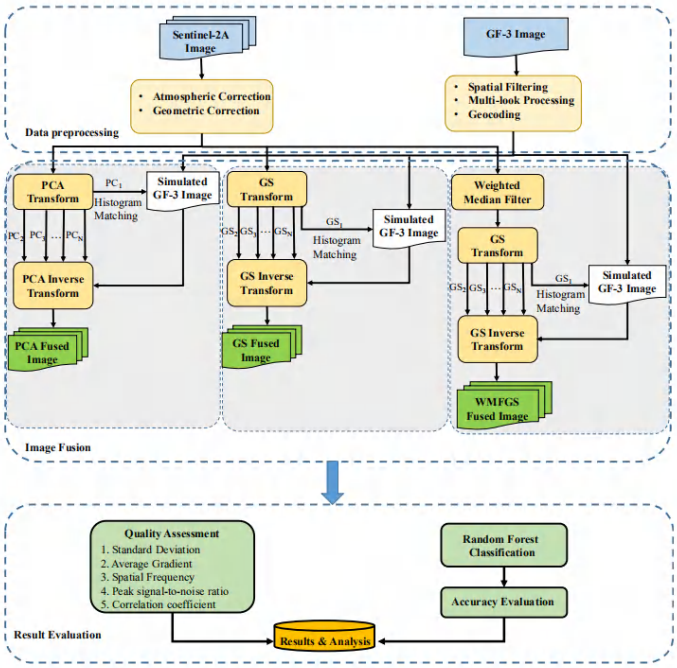


Figure 2.8.: A flow chart to show the process of image fusion[14].

2.1.5. Travel Time Index(TTI)

The Travel Time Index(TTI) is the industry's most used evaluation index of urban congestion degree. TTI is the ratio of the actual travel time and the free flow time. The larger the value, the worse the traffic operation status and the congestion level is generally positive. Related, other abnormal weather conditions (such as rain, snow, fog, etc.) or abnormal road conditions may also affect the value of TTI[15].

The basic idea of TTI is: In the same link in a time slice, TTI = free-flow speed / actual speed, and the calculation method of TTI and Average Speed is shown below:

Assume a set $S = Link_1, Link_2, Link_3, Link_4, ...Link_N$, then:

$$TTI = \frac{\sum_{i=1}^{N} \frac{L_i}{V_i} \cdot W_i}{\sum_{i=1}^{N} \frac{L_i}{V_{free\_i}} \cdot W_i} \tag{2.1}$$

$$Speed = \frac{\sum_{i=1}^{N} L_i \cdot W_i}{\sum_{i=1}^{N} \frac{L_i}{V_i} \cdot W_i} \tag{2.2}$$

Where the size of S is N, the length of link is L_i, the weight of link is W_i, the freeflow of link is V_{free_i}, the realtime speed of link is V_i[15].

2.2. Deep Learning

2.2.1. Convolutional neural network (CNN) and Deep Residual Network(Resnet)

In deep learning, a convolutional neural network (CNN) is a class of deep neural networks. Convolutional networks combine three architectural ideas to ensure some degree of shift and distortion invariance: local receptive fields, shared weights (or weight replication), and sometimes spatial or temporal subsampling[16].

Convolutional neural networks are a promising tool for solving the problem of pattern recognition[17]. For example, the input plane receives an image of a character that is approximately size-normalized and centered, then 6 convolutional matrices use different sets of weights and biases in the convolutional layer will extract different types of local features of the images. In the forwarding process, each unit of a layer receives inputs from a set of units located in a small neighborhood in the previous layer. During this convolutional computation period, neurons in the network can learn to extract elementary visual features such as oriented edges, end-points, corners, and so on. These features are then combined by the subsequent layers to detect higher-order features. After the images forward through all the convolutional layers, several fully connect layers can condense the previous data to the final output layer[16].

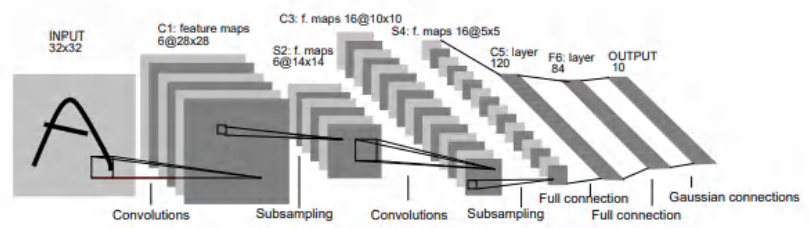


Figure 2.9.: An example of digits recognition by using convolutional neural network[17].

However, when deeper networks can start converging, the problem of degradation of training accuracy has been exposed. Thus, a series of Deep Residual Networks are created to against degradation. Deep Residual Neural Networks (Resnet) let the stacked nonlinear layers fit another mapping of $F(x) := H(x) - x$. The original mapping is recast into $F(x) + x$, which can be realized by feed-forward neural networks with "shortcut connections". The shortcut connections simply perform identity mapping, and their outputs are added to the outputs of the stacked layer, which add neither extra parameter nor computational complexity[18].

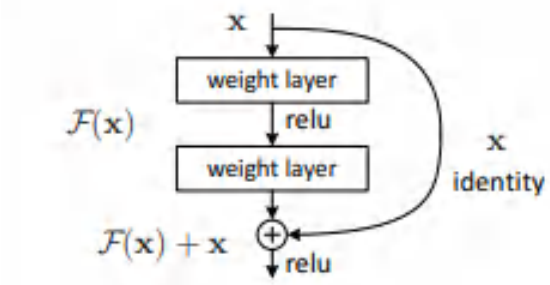


Figure 2.10.: A building block in residual learning[18].

2.2.2. Recurrent Neural Network(RNN) and Long Short Term Memory(LSTM)

Recurrent Neural Networks are a kind of artificial neural network that can use the inputs from previous timesteps to exhibit temporal behavior, which is applicable to analyze time-related, unsegmented, connected, and continuous data[19].

However, Recurrent Neural Networks are limited to look back in time for approximately ten timesteps, which is due to the either vanish or explode of the fed back signal[19]. This issue was addressed with Long Short Term Memory networks (LSTM). Long Short Term Memory is an artificial recurrent neural network that has feedback connections, thus LSTM is capable to learn more than 1000 discrete-time steps[20]. LSTM is widely used in short-term and long-term travel time prediction. A paper written by Irem Islek and Sule Gunduz Oguducu try to predict the travel time for short term and long term using data set from real-time traffic information in New York City[21]. For predictions of both 5 minutes later and 15 minutes later in the task, the error rates of LSTM model are about half of the error rates of another Naïve method: Autoregressive integrated moving average (ARIMA) model.

2.2.3. K-Fold Cross-Validation

Cross-validation is a data resampling method to assess the generalization ability of predictive models and to prevent overfitting. In K-fold cross-validation, first divide the learning set into K equally subsets, then take 1 subset as validation dataset, and the rest of K-1 subsets are training dataset. Repeat the previous process for K times will generate K folds training and validation datasets. Train and validate the K folds datasets will produce K prediction models and generate K results. By averaging or weight averaging the K results, random error due to split of the dataset will be reduced[22].

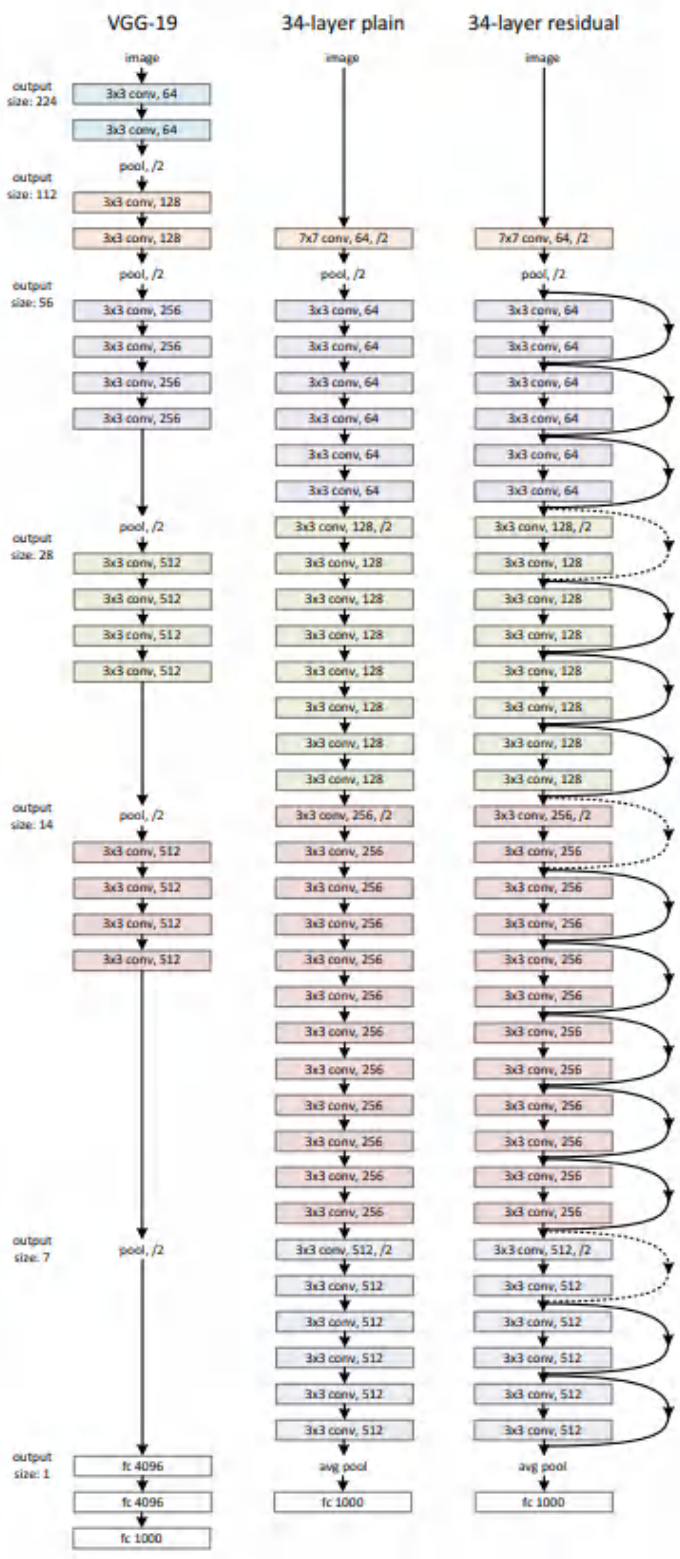


Figure 2.11.: Comparison of Network Architectures between Resnet34 and other CNNs[18].

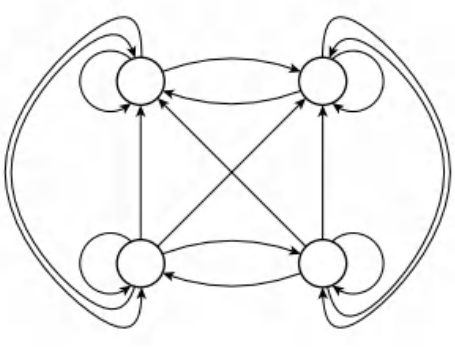


Figure 2.12.: Structure of a fully recurrent neural network (RNN) with self-feedback connections[19].

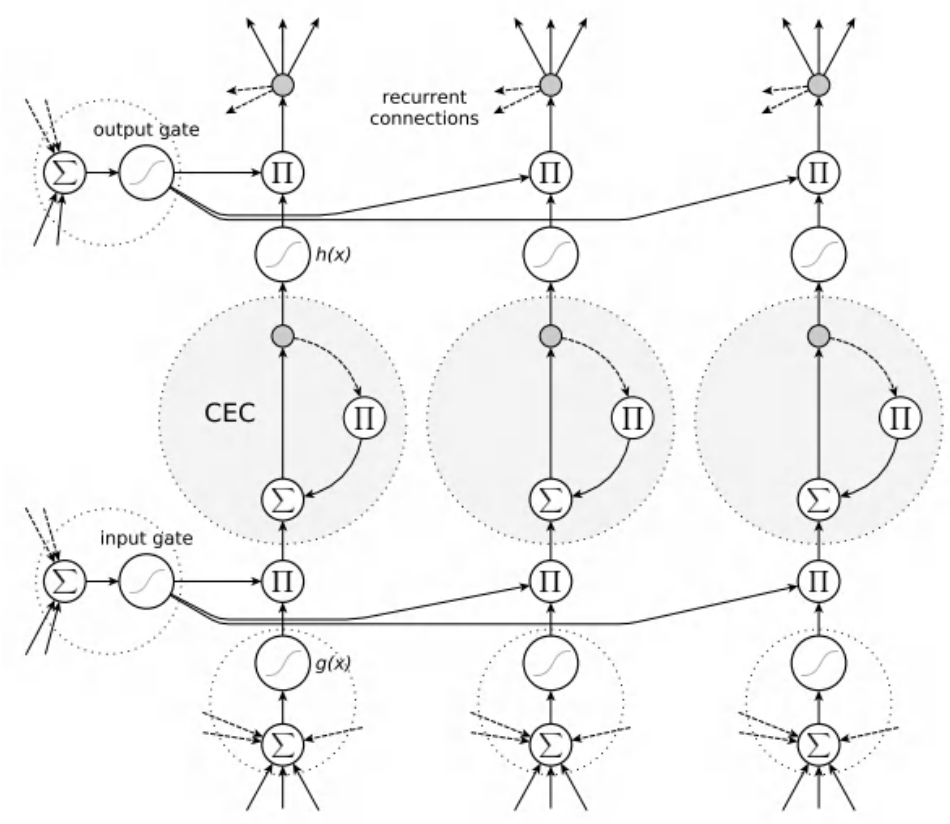


Figure 2.13.: A three-cell LSTM memory block with recurrent self-connections[19].

Test case	Model	Error Rate (MAPE)
5 minutes later	ARIMA	19.4%
5 minutes later	LSTM	9.8%
15 minutes later	ARIMA	23.2%
15 minutes later	LSTM	12.7%

Figure 2.14.: Comparison of results between ARIMA and LSTM models from the research by Irem Islek and Sule Gunduz Oguducu[21].

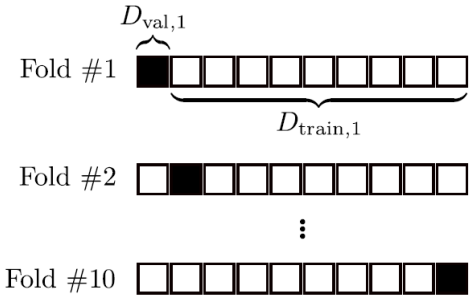


Figure 2.15.: A graphical illustration of dataset splitting in K-fold cross-validation[22].

2.2.4. Learning Rate Reduction

Training a neural network using an algorithm such as error back-propagation usually requires a lot of time on large and complex problems. Such algorithms typically have a learning rate parameter that determines how much the weights can change in response to an observed error on the training set. In one of the researches related to learning rate, to reduce the time consumption during model training and validation, set the learning rate to about 0.01 at first, wait for primarily converge, and then gradually reduce the learning rate to increase the model accuracy during training and validation[23].

In practice, during the phase of model training, we usually use learning rate schedulers to decrease the learning rate[24]. The schedulers will keep the initial learning rate as a constant for the first several steps and then decreases[24]. Theoretically, the decreasing learning rate can infinitely approach zero as the training steps increase. However, one of the widely accepted recommendations states that the learning rate ranging from 1 to 10^{-6} is suitable for most of the networks[24].

Learning Rate	Best Training Epoch	Maximum Generalization Accuracy	Tested Training Epochs
100	3,017	25.63%	5,000
50	3,282	24.70%	5,000
10	1,955	29.21%	5,000
5	4,821	38.66%	5,000
1	1,817	56.14%	1,000
0.5	160	59.96%	1,000
0.1	7	64.86%	1,000
0.05	9	66.62%	1,000
0.01	14	68.71%	1,000
0.005	39	69.04%	1,000
0.001	242	69.48%	1,000
0.0005	473	69.47%	1,000
0.0001	2,252	69.43%	5,000
0.00005	5,913	69.47%	10,000
0.00001	18,595	69.46%	30,000

Figure 2.16.: As the learning rate reduce from 100 to 0.00001, the number of best training epochs decrease first and then increase, and the accuracy converges to about 69.48%[23].

3. Exploratory Data Analysis

3.1. Overview of All Datasets

We will use data from two different sources: GAIA Open Dataset Initiative and the Cadmapper website.

GAIA Open Dataset Initiative, provided by DiDi Chuxing enterprise, provides the academic community with a real-life application use case in the field of traffic forecast, autonomous driving, natural language processing, and so on[25]. Several datasets in GAIA Open Dataset will be introduced and used: Network Dataset, Travel Time Index (TTI) and Average Speed Dataset, and Routing Dataset[26, 27].

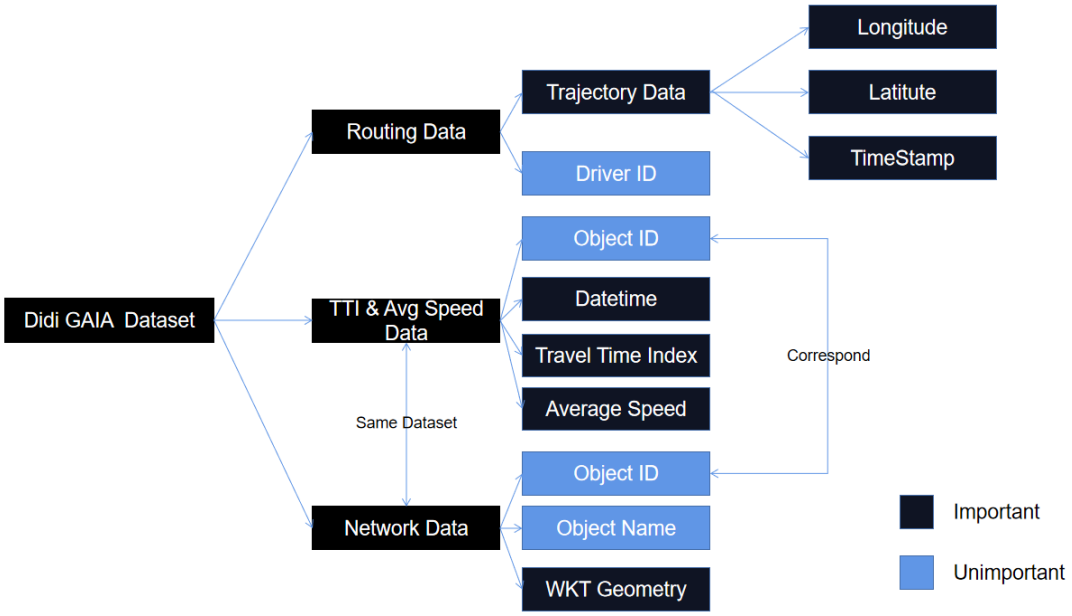


Figure 3.1.: Tree diagram of the structure of Didi GAIA Dataset

As we discussed in chapter 2.1.1, we will use a free 2D CAD file which represents urban layouts of Chengdu City, China, from the Cadmapper website. In our task, the research area is inside Chengdu City, so only part of the file is enough for our research.

3.2. Network Dataset

3.2.1. Fundamental Description of Network Dataset

•Name of Raw Data: 'boundary.txt'
•Size of Data: about 3289 KB
•Time Range: this dataset is irrelevant to time
•Time Interval of Data: this dataset is irrelevant to time
•Space Range: Metropolis in Chengdu, Sichuan province, China

obj_id	obj_name	geom
281863	八里桥路:北站东二路, 三环路	MULTILINESTRING((104.07437 30.71442, 104.0742 30.71483, 104.07
281864	八里桥路:三环路, 北站东二路	MULTILINESTRING((104.07452 30.71324, 104.0745 30.71338, 104.07
281865	二环路西段:广福路, 金牛大道	MULTILINESTRING((104.02746 30.64001, 104.02707 30.64039), (104
281866	二环路西段:金牛大道, 广福路	MULTILINESTRING((104.02536 30.64168, 104.02488 30.64216, 104.0
281867	二环路南段:郭家桥西街, 广福路	MULTILINESTRING((104.079 30.62056, 104.07975 30.62059), (104.0

Figure 3.2.: Demonstration of a part of network dataset in Microsoft Excel

Network data a consist of Object ID, Object Name, and WKT Geometry.

•Object ID, the column with title 'obj_id', is the identification data of the road.
Example: 281931. Which is corresponding to the Object ID in Travel Time Index and Average Speed Data.

•Object Name, the column with title 'obj_name', is the Chinese name of the road, whose detailed information can be further investigated on the database like OpenStreet Map and Google Satellite Map.
Example: (West Section of the Second Ring Road: Guangfu Road, Jinniu Avenue)

•WKT Geometry, the column with title 'geom', is "Well-known text representation of geometry", which can use coordinates to represent various geometric objects, like points, linestrings, polygons, networks and geometry collections, etc.
Example: LINESTRING(104.10583 30.67989,104.10649 30.67871)

3.2.2. Statistical Description of Network Dataset

Network data consists of 1697 rows, among which 21 rows show polygon shapes of administrative regions, and the other 1676 rows show multi-linestring shapes of roads. In this part, only the information of the 1676 individual roads will be analyzed.

	Longitude	Latitude
Minimum	103.17791	30.12575
Maximum	104.86679	31.11004
Range	1.68888	0.98429
Mean	104.0473	30.64799
Median	104.058	30.65033
25 percentile	103.99126	30.59023
75 percentile	104.11593	30.70779
Variance	0.0348	0.0150
Skewness	-0.8763	-0.5037
Kurtosis	5.000	2.3705

Table 3.1.: Statistical description of longitude values and latitude values

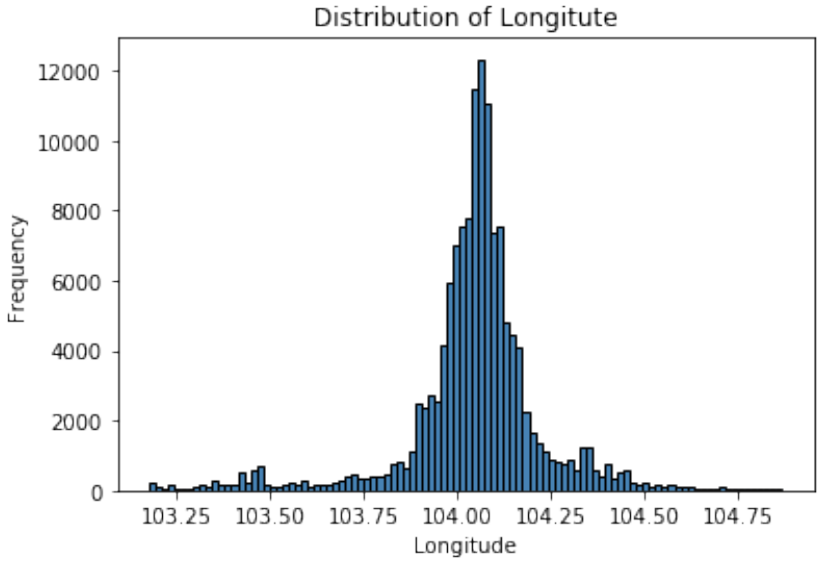


Figure 3.3.: Histogram of longitude values distribution

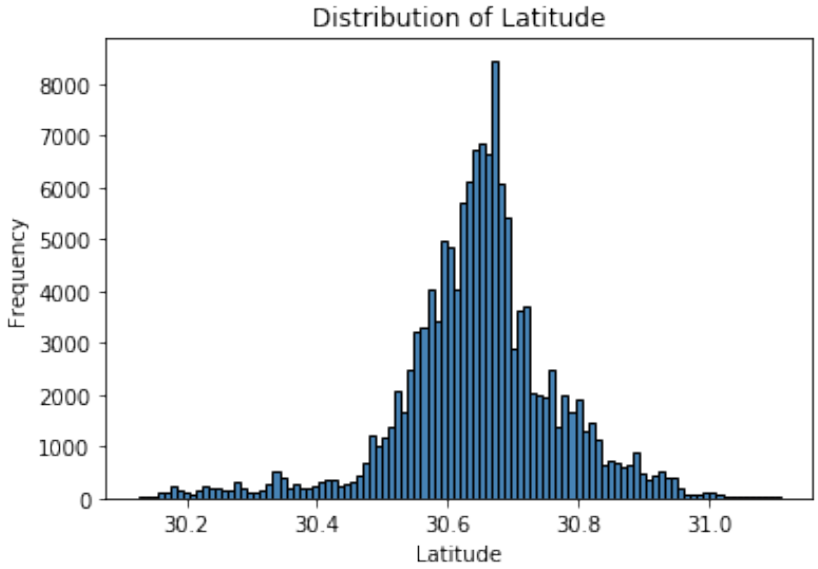


Figure 3.4.: Histogram of latitude values distribution

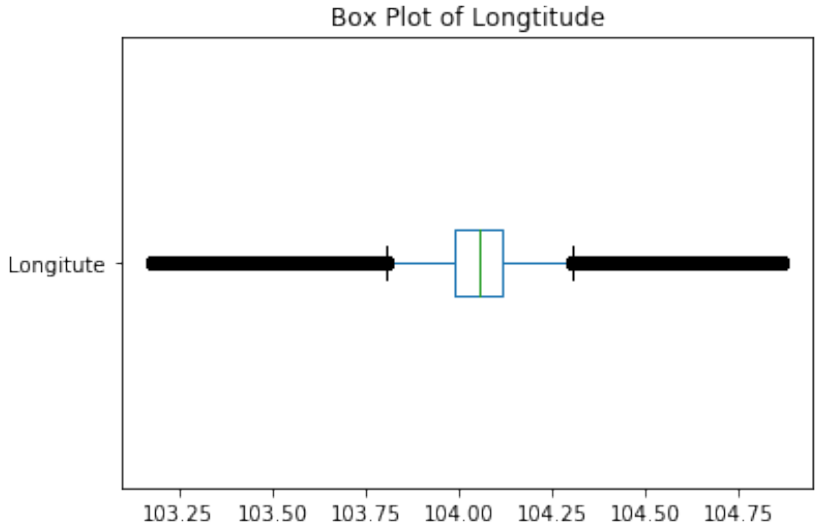


Figure 3.5.: Box plot of longitude values distribution

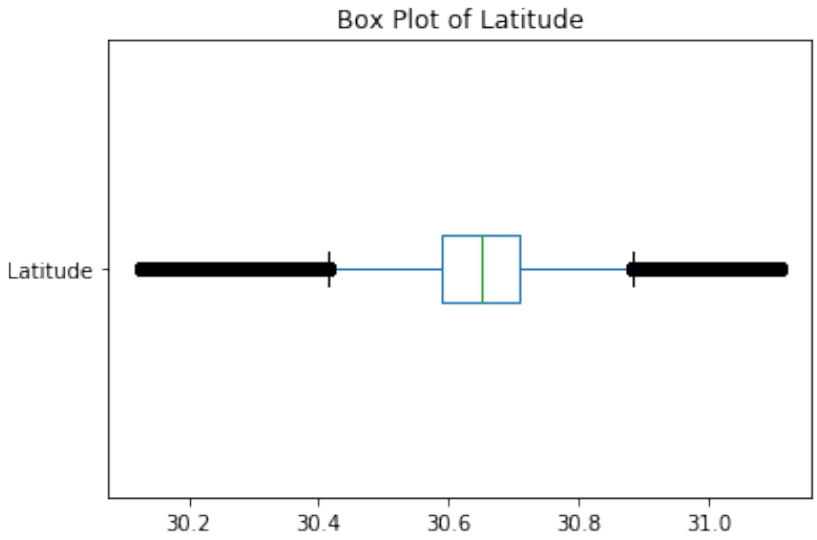


Figure 3.6.: Box plot of latitude values distribution

3.2.3. Discussion of The Statistical Distribution

The distribution of longitude, as well as distribution of latitude, are similar to a normal distribution with low variance, negative skewness>-1, and positive kurtosis>2, which correspond to the highly centralized urban road layouts of the metropolis. Most of the road transportation happens in a relatively small and centralized urban area comparing to the whole Chengdu district.

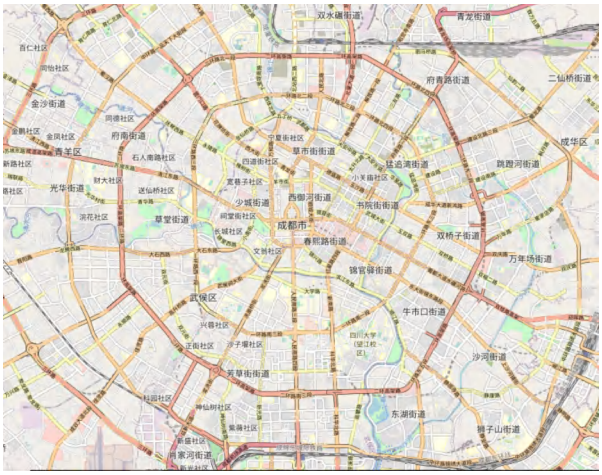


Figure 3.7.: An Open Street Map of Chengdu City shows that the layout of Chengdu is a concentrated patter[28].

3.3. Routing Dataset

3.3.1. Fundamental description of Routing Dataset

•Name of Raw Data: 'chengdushi_1001_1010.zip', 'chengdushi_1010_1020.zip', 'chengdushi_1020_1031.zip', 'chengdushi_1101_1110.zip', 'chengdushi_1110_1120.zip', 'chengdushi_1120_1130.zip'.

•Size of Data: about 24.1GB in total.

•Time Range: From October 1st 2018 to December 1st 2018.

•Time Interval of Data: Several seconds. Time interval is fixed for one identical trajectory but may vary from different trajectories.

•Space Range: restricted district by four GCJ-02 coordinates points ([latitude, longitude]): [30.727818,104.043333], [30.726490,104.129076], [30.655191,104.129591], [30.652828,104.042102] inside Chengdu, Sichuan province, China.

Driverid_1	Driverid_2	Trajectory
4ecea6b8473789e1fdbfea71adfb2451	8ac962c8b85724f4602dbc9e50c3ff46	[104.12226 30.67012 1539952150,
3018c0c10f32ec1aab592168bb6f6787	7535711890e34f9476ad616fba2648e4	[104.05973 30.68469 1539949303,
876853121ec96be317b065c286081976	cf9d01b5bf5de4415c3a56f0c6815b53	[104.09237 30.67939 1539949359,
414b5e790bdc74bc23ce18694ce04909	065bc24b5194fa06b7d87c592c4d9582	[104.09797 30.65295 1539949558,
bb53ed3292cbccc4faca7a20effd221b	f8f1a4a9f0ccc3af0c7a5db7b91e4fc5	[104.04503 30.70138 1539949951,

Figure 3.8.: Demonstration of a part of routing dataset in Microsoft Excel

Routing Data consist of Driver ID and Trajectory Data.

•Driver ID, the column with title 'Driverid_1' and 'Driverid_2', are the desensitization of drivers' personal information.

Example: 4ecea6b8473789e1fdbfea71adfb2451.

•Trajectory Data, the column with title 'Trajectory', consist of 3 parts: longitude, latitude and timestamp in order.

Example: 104.12226 30.67012 1539952150, 104.12226 30.67012 1539952160.

1. Longitude: longitude data based on GCJ-02 coordinate system.

Example: 104.12226.

2. Latitude: latitude data based on GCJ-02 coordinate system.

Example: 30.67012.

3. Timestamp: the Unix time when the longitude and latitude data is recorded.

Example: 1539952150.

3.3.2. Statistical Description of The Routing Dataset

Since the positional data has been analyzed in Network Data, we will compute the driving speed data from Trajectory Data and analyze it. The formula of driving speed computation is:

$$Speed_i = \frac{\sqrt{((Long_{i+1} - Long_i)^2 + (Lati_{i+1} - Lati_i)^2)}}{(Time_{i+1} - Time_i)} \cdot 111.19 \cdot 3600 \qquad (3.1)$$

Where $Long_i$, $Lati_i$ and $Time_i$ are values of longitude, latitude and timestamps in the i^{th} 3-element tuples of Trajectory Data. $Speed_i$ is the average speed during the time $Time_{i+1} - Time_i$, which are indicated in chapter 3.3.1. Since $Time_{i+1} - Time_i$ are about several seconds, so $Speed_i$ can also be regarded as the instantaneous speed at $Time_i$.

According to U.S. Geological Survey, the distances of one degree of longitude or latitude vary. For example, at 38 degrees North latitude, one degree of latitude equals approximately 111 kilometers, and one degree of longitude equals approximately 87.9 kilometers. In terms of convenient calculation, we use one degree of either longitude or latitude equals to approximately 111.19 kilometers[29].

By using equation (3.1), we calculate the values of speed and analyze the data. Because of the huge size of the raw data, we will analyze 122635 values calculated from 600 routes on October 8th 2018 as an example:

	Speed
Minimum	0.02
Maximum	14913.28
Range	14913.26
Mean	26.69
Median	42.48
25 percentile	10.82
75 percentile	44.03
Variance	4113.20

Table 3.2.: Statistical description of speed values

To eliminate the influence of outliers, we only plot the individual speed from 1 percentile to 99 percentile.

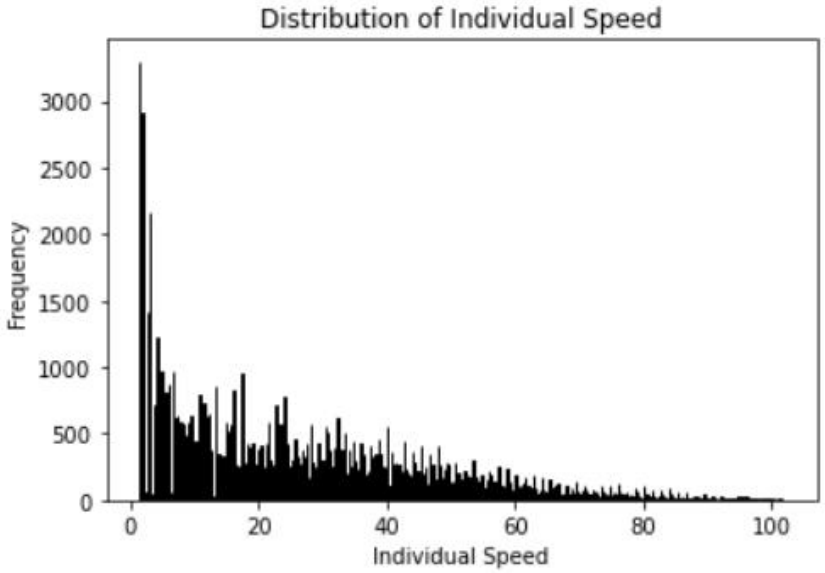


Figure 3.9.: Histogram of speed values distribution

3.3.3. Discussion of The Statistical Distribution

The values of individual speed have two abnormal features: One is some values reach over 10000, which is impossible in reality. Maybe because the raw data itself is not accurate enough, but these outliers can be regarded as noises since the amount of them is not large. Another is a big proportion of values are nearly zero, but this phenomenon usually happens when vehicles are congested or waiting for traffic lights.

3.4. Travel Time Index(TTI) and Average Speed Dataset

3.4.1. Fundamental description of TTI and Average Speed Dataset

- Name of Raw Data: 'road.zip'
- Size of Data: about 901840 KB
- Time Range: From 00:00:00 January 1st 2018 to 23:50:00 December 31st 2018
- Time Interval of Data: fixed 10 minutes
- Space Range: Metropolis in Chengdu, Sichuan province, China

TTI and average speed dataset consist of Object ID, Datetime, Travel Time Index, and Average Speed.

- Object ID, the column with the title 'Object_ID', is the identification data of the road. Which is corresponding to the Object ID in Network Data.

Example: 281931.

•Datetime, the column with the title 'Datetime' is the date and time when the specific road has the indicated travel time index and average speed.

Example: 2018/10/19 20:00:00.

•Travel Time Index(TTI), the column with the title 'TTI', is a link-weighted ratio of actual travel time and free-flow time. High TTI indicates the severe congestion situation of a specific road.

Example: 2.45013.

•Average Speed, the column with title 'Avg Speed', is the average speed of all the vehicles in the specific road at the specific time.

Example: 14.7419.

Object ID	Datetime	TTI	Avg Speed
283404	2018/1/1 0:00	1.39778	39.389
283442	2018/1/1 0:00	1.07396	82.3449
283002	2018/1/1 0:00	1.09723	27.7525
282048	2018/1/1 0:00	1.74019	15.2246
283424	2018/1/1 0:00	1.32598	35.5202

Figure 3.10.: Demonstration of a part of TTI and average speed dataset in Microsoft Excel

3.4.2. Statistical Distribution of TTI and Average Speed Dataset

The Travel Time Index(TTI) and Average Speed Data is recorded from 00:00:00 January 1st, 2018 to 23:50:00 December 31st, 2018, whose time range is larger than that of Routing data, which is from 00:00:00 October 1st, 2018 to 23:50:00 November 30th, 2018. To prepare for multispectral image data fusion in a later part, we only select the Travel Time Index(TTI) and Average Speed Data from 00:00:00 October 1st, 2018 to 23:50:00 November 30nd 2018, a totally of 61 days.

First, Investigate the completeness of selected data. Theoretically, if the selected data is completed, all the roads will have identically $61 * 24 * 6 = 8784$ sets of TTI and Average Speeds data. However, by statistical data analysis, 1668 roads have been investigated with 8783 sets of TTI and Average Speeds data recorded in maximum, and 7 sets of TTI and Average Speeds data recorded in minimum.

For the top 3 frequencies of the number of data recorded: 447 roads have 8783 sets of TTI and Average Speeds data, 79 roads have 8782 sets of TTI and Average Speeds data, and 42 roads have 8781 sets of TTI and Average Speeds data.

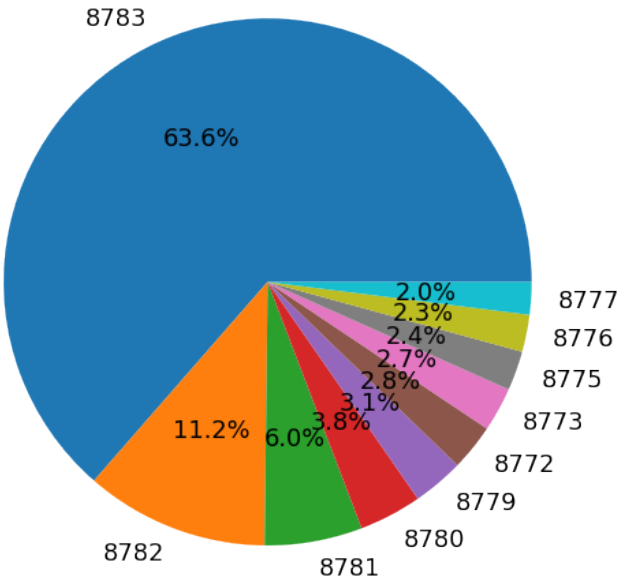


Figure 3.11.: Pie Chart of top 10 frequencies of the number of data recorded

Based on the previous analysis, TTI and Average Speeds data is uncompleted, which means the data has some missing values. And the statistical description of both TTI And Average Speeds is shown below:

To eliminate the influence of outliers, we only plot the TTI values and Average Speed values from 1 percentile to 99 percentile, that is, TTI values within the range [0.9183, 3.6549], and Average Speed values within the range [10.3769, 95.3101].

Urban transportation is generated by human activities, and residents lived in urban district follows a generally fixed lifestyle: On weekdays, most of the people go to work in the morning, and go back to home in the afternoon or at night, with possible recreational activities and other stuff. On weekends, most of the people stay at home or do recreational activities and other stuff, with some people may also go to work like weekdays. Thus the scheme of urban transportation will follow a periodic distribution. We investigate the relationship between average speed, TTI, and time by plotting the data on road No. 283509 as an example, from October 1st, 2018 to October 3rd, 2018.

From the graphs above, there is a negative correlation between average speed and Travel Time Indexes, and both of them are approximately periodic for days. Besides, there are some outliers points whose locations are far from the tendency line.

	Travel Time Index	Average Speed
Minimum	0.131779	0.1
Maximum	507.394	150.0
Range	507.262221	149.9
Mean	1.4474	33.552
Median	1.3152	31.5732
25 percentile	1.15855	24.9988
75 percentile	1.53686	38.8447
Variance	0.7944	197.347
Skewness	2.10522	1.57466
Kurtosis	6.49817	4.75767

Table 3.3.: Statistical description of TTI and Average Speed values

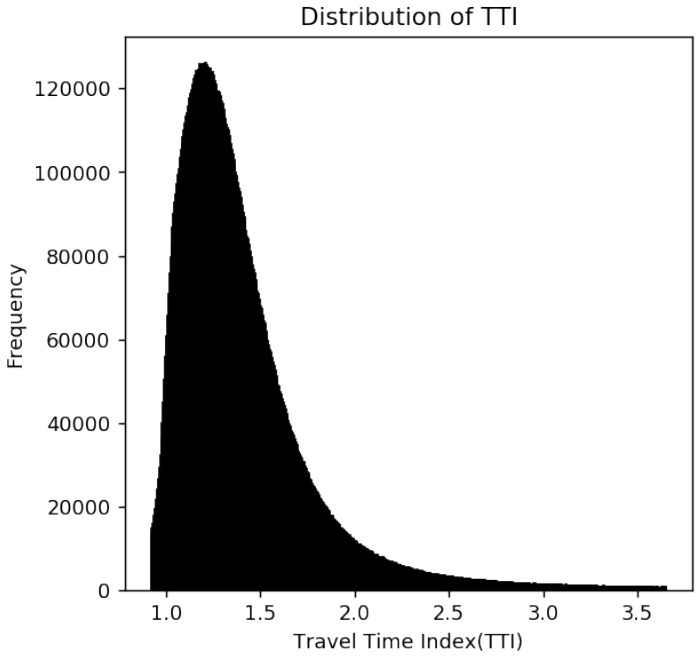


Figure 3.12.: Histogram of Travel Time Indexes values distribution

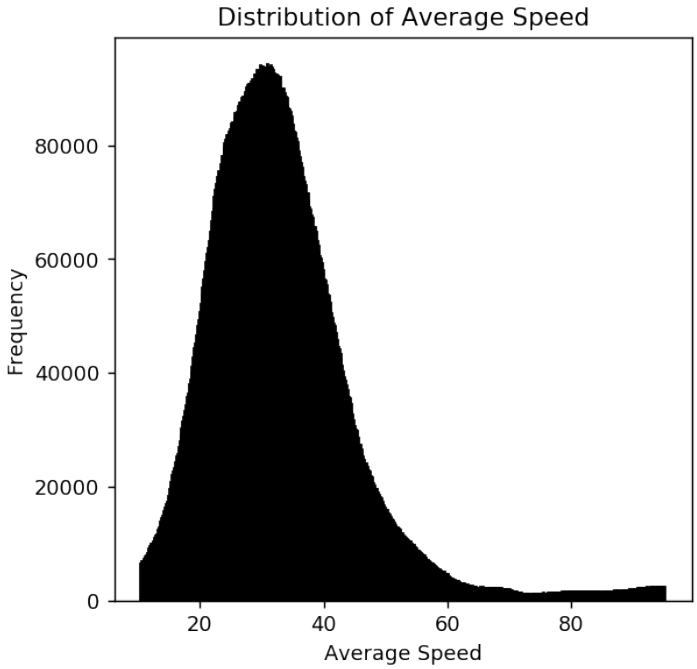


Figure 3.13.: Histogram of Average Speed values distribution

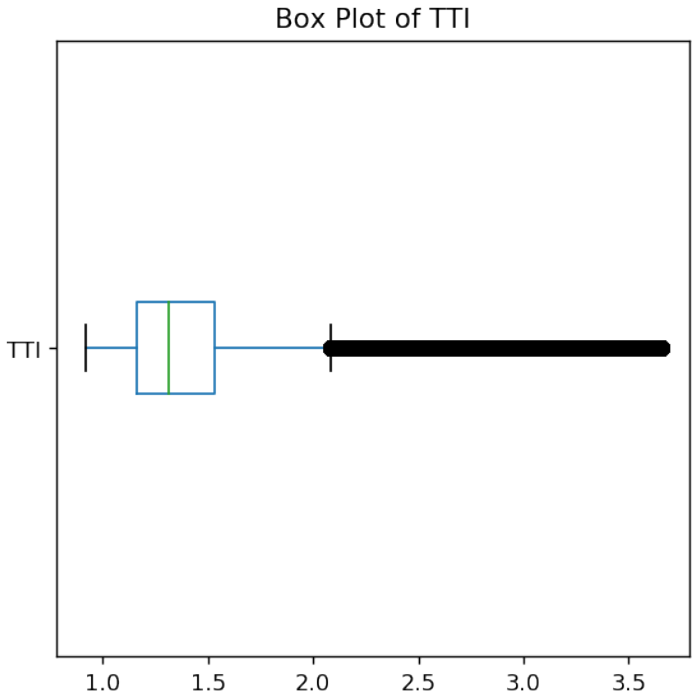


Figure 3.14.: Box plot of Travel Time Indexes values distribution

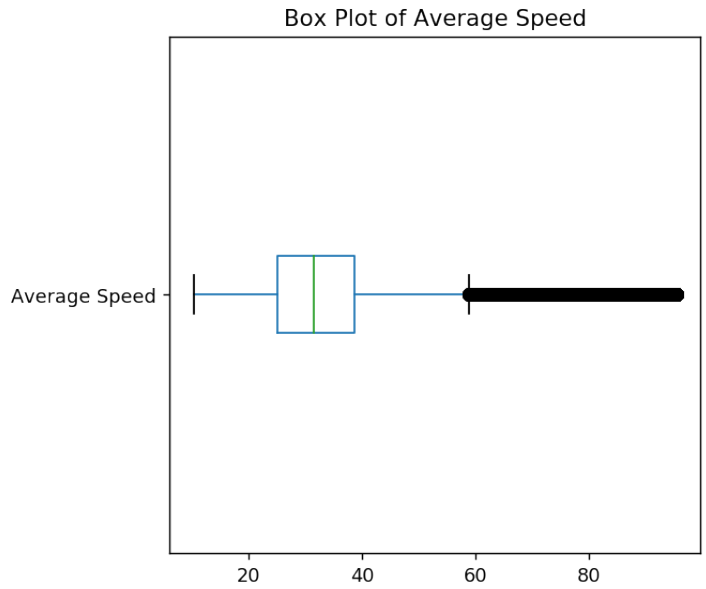


Figure 3.15.: Box plot of Average Speed values distribution

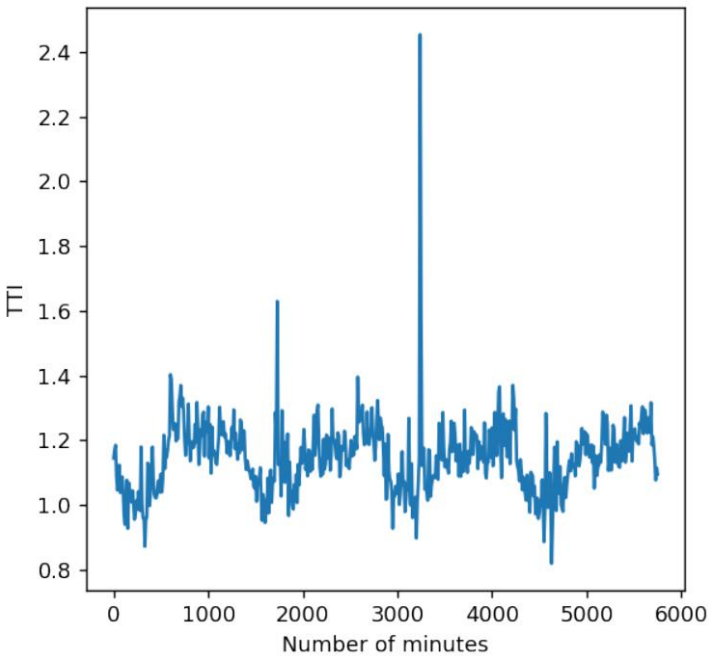


Figure 3.16.: TTI vs time on road No.283509 from Oct. 1st 2018 to Oct. 3rd 2018

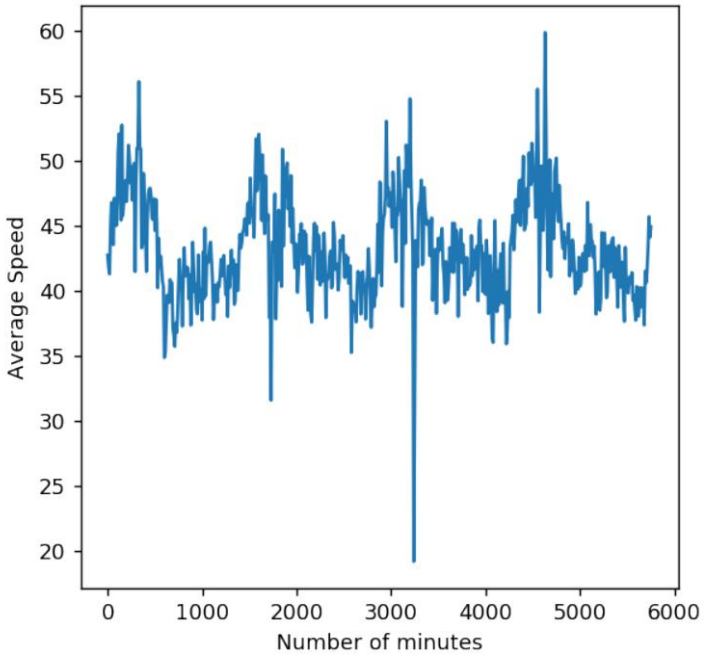


Figure 3.17.: Average speed vs time on road No.283509 from Oct. 1st 2018 to Oct. 3rd 2018

3.4.3. Discussion of The Statistical Distribution

Since the TTI and Average Speeds data have some missing values, we need to fill in the missing values. One of the methods is using average values of existing data to fill in the missing values. However, for roads that only have few existing values (E.g. only 7 values existed in road No.283315), it's impossible to fill in the missing values, thus the traffic forecast of these roads using deep learning methods is invalid.

Distribution of Travel Time Indexes, as well as distribution of Average Speed, both have positive skewness > 1.5 and positive kurtosis > 4.7. The distribution of these data is similar because the values of Travel Time Indexes are proportional to the average speed, as the definition formula of Travel Time Indexes indicated. From 25 percentile to 75 percentile distribution of average speeds are within the range of 25km/h to 38.8km/h, because there is a similar speed regulation on the urban road in most of the Chinese metropolis.

The definition of Travel Time Indexes indicates that Travel Time Index is a value that always larger or equal to 1. However, some of the Travel Time Indexes values are less than 1. Theoretically, the speed of vehicles can be infinite if they are in free flow. However, in reality, free-flow speed is defined by a maximum allowed speed by transport regulations. While some of the drivers exceed the speed limit by the regulations, Travel Time Indexes are less than 1. Thus, transportation regulators

can track and punish the speeding drivers by monitoring the values of Travel Time Indexes.

4. Methodology

4.1. Assumption and Pre-settings

4.1.1. Reduction of Precision

When transforming the textile data to graphical data, minimum precision should be assigned. In both the Network Dataset and Routing Dataset, the precision of both longitude and latitude is 0.00001 degree. However, reducing the precision to 0.0001 degrees will significantly decrease the computation cost. Thus in this research, the minimum precision of distance increase from 11.119 meters to 111.19 meters.

4.1.2. Ignore the Width of Road

Generally, the width of a single lane is less than 4 meters[30]. Thus most of the width of roads, avenues, and highways is less than 50 meters. Under the new 111.19 meters precision, the width of roads can be ignored, which significantly decreases the computation cost. In our macroscopic analysis, the research area covers a large part of metropolis districts, contrasting sharply to small parts of single roads. Under the macroscopic view, the roads look like single lines with no width. Besides, in the Network Dataset, the roads are represented by multiple lines.

4.1.3. Fixed Free Flow Speed

In chapter 3.4.3, we conclude that the free flow speed is defined by traffic regulations. According to WHO, research on effective speed management indicates that the speed limits on urban roads should not exceed 50 km/h[31]. Although the speed limits on the urban road can vary in different cities, we can assume that the free flow speed in a designated city is fixed.

4.1.4. Restriction of Research Area

Base on the basic illustration of datasets in chapter 3, compared to network data, the routing data is restricted in a smaller area. To ensure the data fusion from different sources, we only use data inside the small area of routing data, which is:

- Lower bound of longitude: 104.0402.
- Lower bound of latitude: 30.6516.
- Upper bound of longitude: 104.1298.
- Upper bound of latitude: 30.7284.

We assume the precision (0.0001 degree of longitude and latitude) is equal to one pixel, which is the minimum unit of the image. Then the resolution of pictures will be:

- Width: 896 pixels
- Height: 768 pixels

4.2. Featured Engineering

4.2.1. Geographical Images

4.2.1.1 Geographical Image Description

One Geographical Picture will represent The geographical landscape of the research area.

The geographical landscape will affect the urban layout of cities, thus affect the accessibility of different locations. For example, both the Yangtze River and Jiangling River flow across Chongqing city, which generate a special distribution of roads. Most of the urban area of Chongqing city distributes along the riversides.

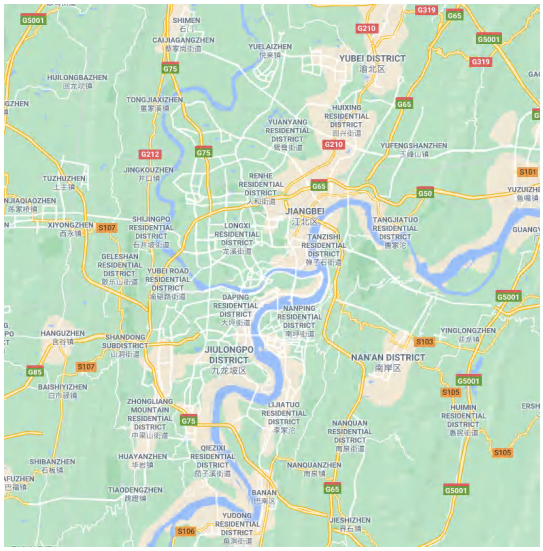


Figure 4.1.: Google Map of Chongqing City, China[32]

When analyzing macroscopic graphical traffic data in a metropolis, the size of both graphical network data and trajectory data are often very huge. So sometimes we need to divide the raw graphical data into several pieces of parts, and then apply the computer vision to the parts[33]. Thus, the application of geographical data can help the neural network to memorize the locations of areal parts.

4.2.1.2 Generation of Geographical Images

The geographical 2D CAD picture of Chengdu City, called "chengdu.dxf", is downloaded from the Cadmapper website[8]. Because of the research area restriction in Chapter 4.1.4, only the restriction area of the picture can be used, thus we need to extract the restriction area from the raw picture.

However, the coordinate system used in the CAD picture is different from GCJ-02, thus we need to establish a mapping of the coordinate system between the coordinate system of GCJ-02 and that of the CAD picture. The processes are:

A. Open both 'chengdu.dxf' in AutoCAD software and an inquiry website of GCJ-02 positional coordinates in Gaode AI platform[34].

B. Localize a series of easily recognized landmarks on both of the AutoCAD software and Gaode AI platform, and record the coordinates individually.

C. Use linear regression to find the mapping relationship between the two coordinates.

D. Localize the restriction area in the CAD file and extract it as individual geographical data.

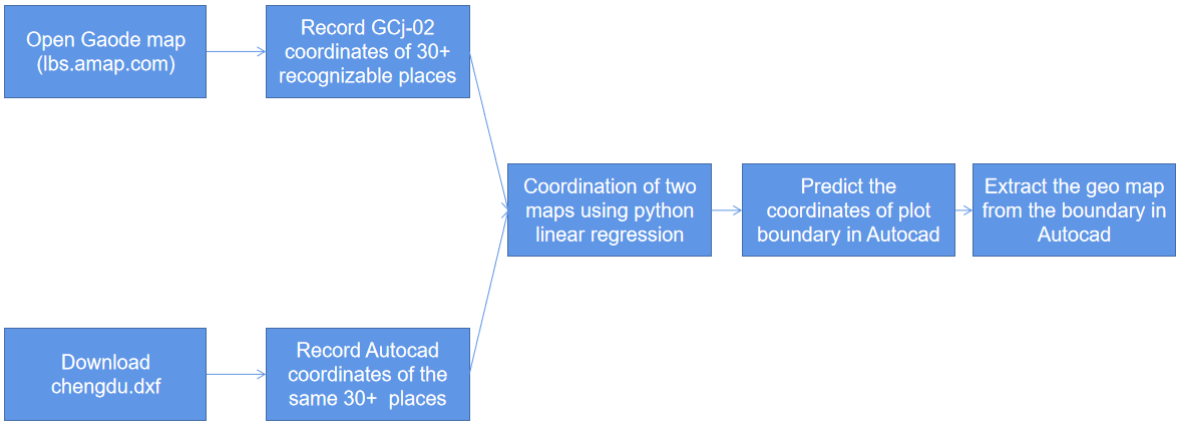


Figure 4.2.: Flow chart of Geographical Images generation

In step B, the corresponding coordinates of landmarks are shown in appendix.

In step C, the linear regression result is shown below:

- X_intercept: -9869890.84
- X_slope: 95306.65
- Y_intercept: -3379097.01
- Y_slope: 111277.64

So the mapping relationships between the coordinates of CAD graphs and MAP graphs are:

$$CAD_X = 95306.65 \cdot MAP_X - 9869890.84 \tag{4.1}$$

$$CAD_Y = 111277.64 \cdot MAP_Y - 3379097.01 \tag{4.2}$$

Using the function above and the coordinates of boundaries in MAP in chapter 4.1.4, the coordinates of boundaries in the CAD is calculated as following:

- Lower bound of longitude: 45831.91884367.
- Lower bound of latitude: 31740.6771768.
- Upper bound of longitude: 54371.39453595.
- Upper bound of latitude: 40286.79986843.

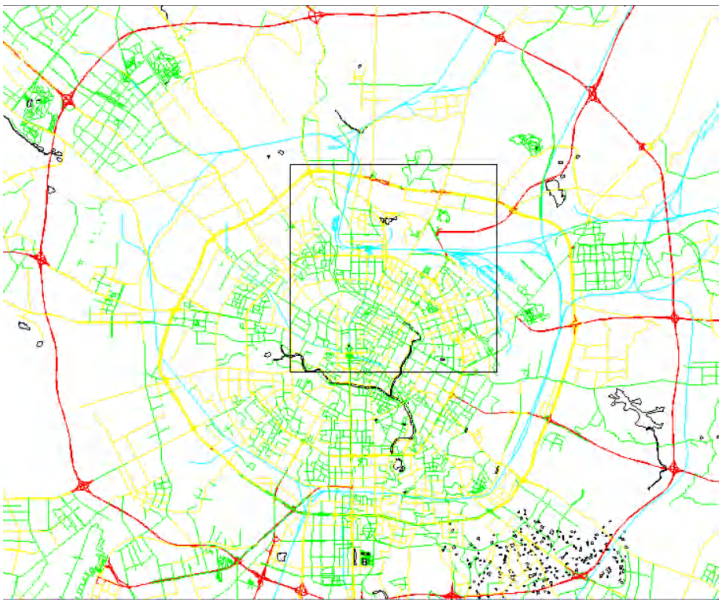


Figure 4.3.: Boundaries in AutoCAD shown by the black rectangle

Geographical Images Demonstration

At last, the geographical picture has been generated, with the different colors represent different kinds of roads or landscapes.

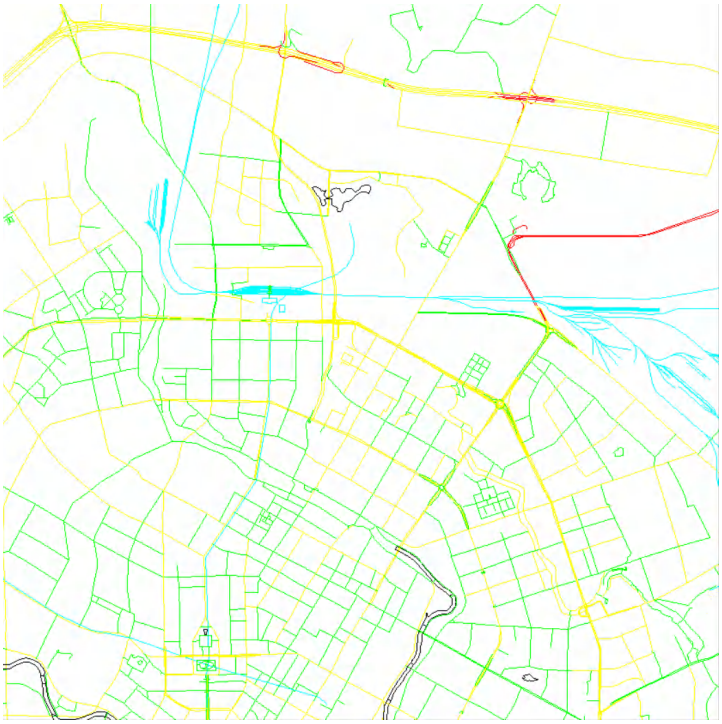


Figure 4.4.: Demonstration of Geographical Image

Due to the unavoidably operational error in the whole process, the initial size of the Geographical Image is $1042 * 1043$. In the data fusion process below, the image will be resized to the required resolution.

4.2.2. Roadmap Images

4.2.2.1 Roadmap Images Description

One roadmap image will represent for:

- The layout of the urban road in Chengdu City
- The average speed of vehicles on the road

The time interval of every two successive images is 10 minutes. For instance, if one roadmap image represents 20:00, then the two successive images represent 20:10 and 20:20.

The roadmap image will be similar to the heatmap image. In the heatmap image depicted below, most of the images are in darkness, which indicates that the speed of vehicles in a pure dark area is 0. As the speed of vehicles goes higher, the area of vehicle flow goes brighter[35].

Figure 4.5.: A heatmap in Chengdu City from Strava Heatmap website[35]

In contrast, most of the Roadmap images will be white. In pure white area, the speed of vehicles is 0. As the speed of vehicles goes lower, the area of vehicle flow goes darker.

4.2.2.2 Decision of the width of plotting lines

In the python matplotlib plot command, another parameter besides the 'color' value that mainly affects the pixel values is the 'linewidth' value. The 'linewidth' value will determine the width of plotting lines, which represent the width of the road. To figure out the combined effect of these two values onto pixels, we plot a linewidth = 1, slope = -1 line and check the pixel of the crossing line.

```
x = [0, 1]
y = [1, 0]
plt.rcParams['figure.dpi']=100
plt.rcParams['figure.figsize']=(1, 1)
plt.plot(x, y, color = 'black', linewidth = 1.0)
plt.axis('off')
plt.savefig('test0.png')
```

Figure 4.6.: Plot a line with parameters: linewidth = 1, slope = -1 in Jupyter Notebook

By looking into the value of pixel, there is a tuple: (229,36,31,224)

The mathematical interpretation of the tuple is that: part of the line intersect the four pixels and reduce the values of the four pixels.

```
from PIL import Image
import numpy as np
import os
img = Image.open('test0.png')
ig = np.array(img)
ig[:,50,0]
```

```
array([255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
       255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
       255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
       255, 255, 255, 255, 255, 255, 255, 255, 229,  36,  31, 224, 255,
       255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
       255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
       255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
       255, 255, 255, 255, 255, 255, 255, 255, 255], dtype=uint8)
```

Figure 4.7.: Show values of pixels at the 50th layer in Jupyter Notebook

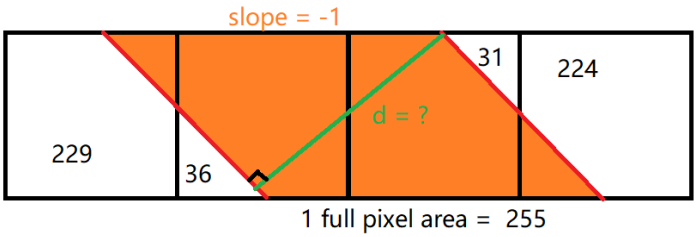


Figure 4.8.: Mathematical interpretation of the tuple

The process of geometric calculation is shown below:
Assume a is the width of one-pixel area, the area of one full pixel is 255:

$$a^2 = 255 \tag{4.3}$$

The area of orange part have two representations:

$$\sqrt{2}a \cdot d = 255 \cdot 4 - 229 - 36 - 31 - 224 \tag{4.4}$$

The result is about $d = 1.386a$, which means linewidth = 1 represent for 1.386 pixel occupation (or 150 meters).

Assuming that an urban road has 4 lanes, the width of the road will be about 15 meters. When plotting this road, the linewidth will be set to 0.1.

However, in deep learning computer vision, a small linewidth value will make the neural network hard to recognize the tiny characteristics in the Roadmap images. In this dilemma, we increase the linewidth to 0.3. On the one hand, the widths of the roads have been ignored in chapter 4.1.2, so the change of the linewidth is durable. On the other hand, linewidth = 0.3 represents for 50 meters actual road width, which is large but not exaggerated.

4.2.2.3 Generation of Roadmap Images

The process of Roadmap images generation is:

A. Extract the part of both TTI and Average Speed Data and Network Data inside the research area indicated in Chapter 4.1.4.

B. Concatenate the two data into one data based on the same road ID, which includes the information of road ID, Datetime, speed, TTI, and the WKT geometrical coordinates of roads.

C. Use the median of existing values to fill in the missing values in the new concatenated file.

D. Transform the longitude values and latitude values to the x and y coordinate of a point, where x and y are positional values concerning width and height in the images.

E. Transform the speed to the value of the pixel, which is set to be an integer ranging from 0 to 255. The transforming formula is Value of pixel = 255 - integer of speed. Where the integer of speed is the integer rounded value of speed.

F. Use a python package, matplotlib, to plot the image.

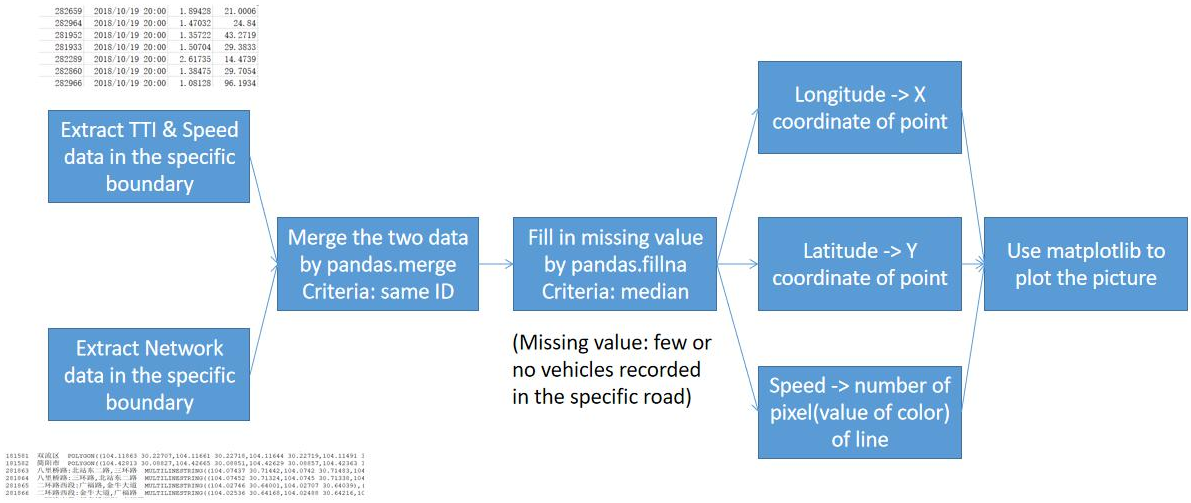


Figure 4.9.: Flow chart of Roadmap Images generation

4.2.2.4 Roadmap Images Demonstration

We generate 3603 Roadmap images from 00:00:00, October, 8th 2018 to 00:20:00, November 3rd, 2018, about 25 days. 144 images in each day with 10 minutes time interval, and additional three images for buffer.

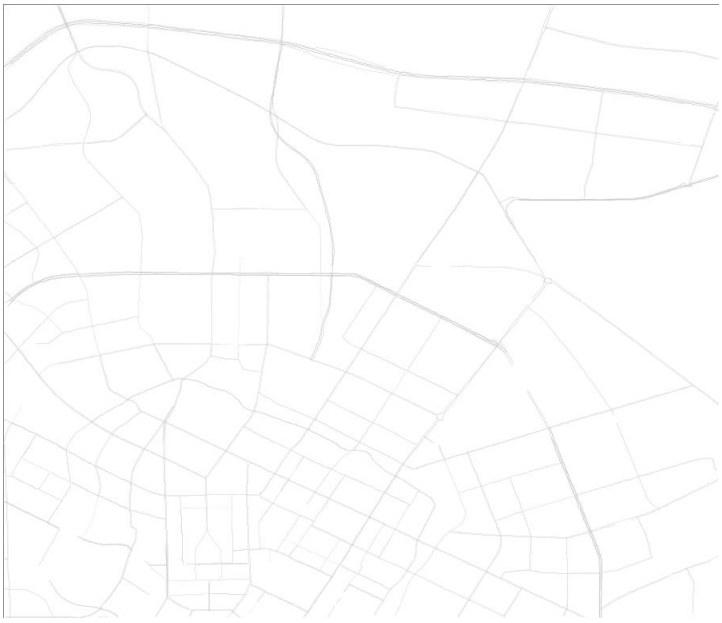


Figure 4.10.: Roadmap image of November 8th, 2018 at 03:00:00 (size 138KB)

Figure 4.11.: Roadmap image of November 8th, 2018 at 09:00:00 (size 120KB)

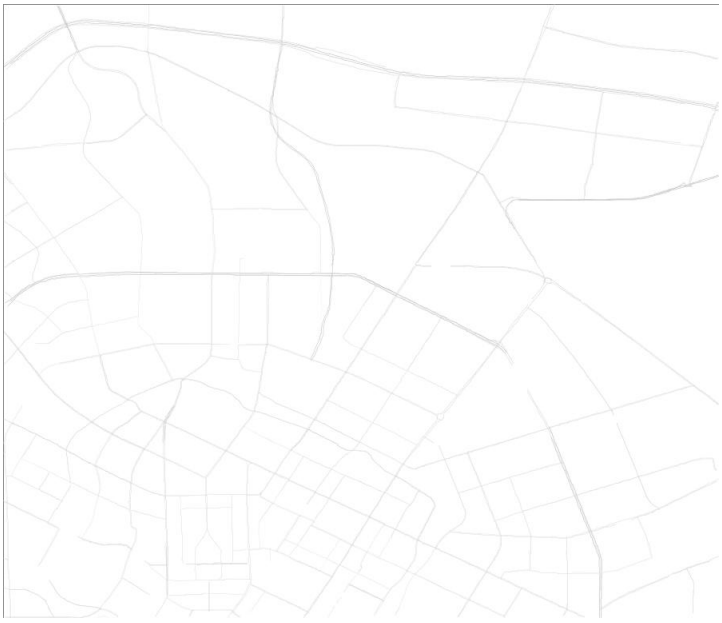


Figure 4.12.: Roadmap image of November 8th, 2018 at 15:00:00 (size 128KB)

Figure 4.13.: Roadmap image of November 8th, 2018 at 21:00:00 (size 129KB)

Since all Roadmap Images are using the same network data, so it is hard to see the difference between the four Roadmap Images. However, the sizes of the images are different, which means the images are not the same.

4.2.3. Trajectory Images

4.2.3.1 Trajectory Images Description

One Trajectory image will represent for:

- The trajectory of the drivers.
- The speed of the drivers.

The time interval of every two successive images is 10 minutes. For instance, if one Trajectory image represents 20:00, then the two successive images represent 20:10 and 20:20.

However, the trajectories of drivers are recorded two minutes before the time, and the duration is 4 minutes. For example, in the image represent 20:00, the trajectories of drivers are recorded since 19:58 and terminated at 20:02.

In the Trajectory images, the darker area corresponds to higher speed, and the pure white area means 0 speed, which is identical to Roadmap images. Additionally, denser lines mean more drivers driving on roads.

4.2.3.2 Decision of Trajectory Recording Duration

Based on the data description in Chapter 3, the Trajectory Data has a smaller time interval than that of TTI and Average Speed Data. Thus the Trajectory Data can be regarded as continuous concerning time. However, for data fusion, the continuous Trajectory Data should be transformed to a discrete Trajectory image with 10 minutes time interval.

For the decision of trajectory recording duration, the duration is an integer value ranging from 2 to 600 with the unit of seconds. Consider two extreme conditions for images represent for 00:10:00:

- 600 seconds duration: e.g. 00:05:00-00:15:00. Advantages: Keep the time-related vehicle transport continuous, and there is no lost of raw data.
- 2 seconds duration: e.g. 00:09:59 - 00:10:01. Advantages: High relevance to the Roadmap image, since TTI and Average Speed values are recorded simultaneously at 00:10:00.

At last, we use a compromise interval of 4 minutes duration, as indicated above.

4.2.3.3 Generation of Trajectory images

The process of Trajectory images generation is:

A.Use Trajectory Data inside the 4 minutes trajectory recording duration, by filtering the 'Timestamp' data.

B.Transform the longitude values and latitude values to the x and y coordinate of a point, where x and y are positional values concerning width and height in the images.

C.Compute speed by Euclidean distance of (x,y) tuple, and time interval. The formula is indicated in equation (3.1).

D.Discard missing values and modify error values like 633.78 km/h (in Chapter 3.3.2) to 150 km/h(maximum speed in Chapter 3.2.2)

E.Transform the speed to the value of the pixel, which is set to be an integer ranging from 0 to 255. The transforming formula is: Value of pixel = 255 - integer of speed. Where the integer of speed is the integer rounded value of speed.

F.Use a python package, matplotlib, to plot the image. Similar to Chapter 4.2.2, we use linewidth = 0.03 rather than 0.01.

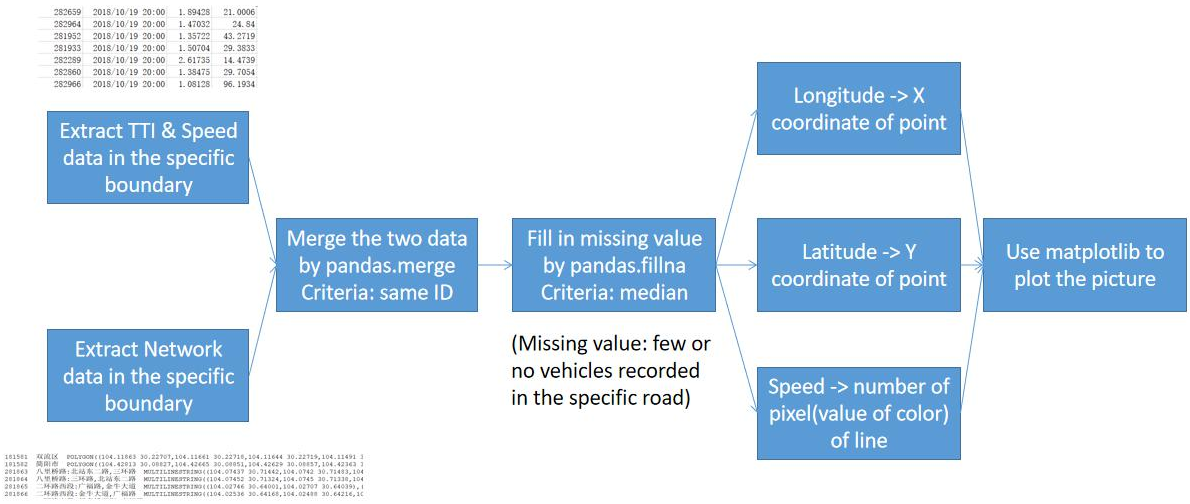


Figure 4.14.: Flow chart of Trajectory Images generation

4.2.3.4 Trajectory Images Demonstration

We generate 3603 Trajectory images from 00:00:00, October, 8th 2018 to 00:20:00, November 3rd, 2018, about 25 days. 144 images in each day with 10 minutes time interval, and additional three images for buffer.

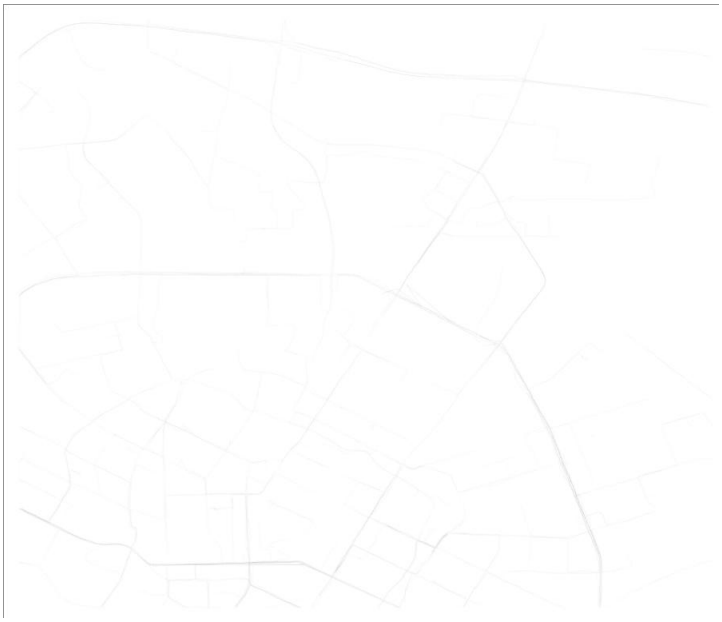


Figure 4.15.: Trajectory image of November 8th, 2018 at 03:00:00, whose trajectory record duration is from 02:58:00 to 03:02:00

Figure 4.16.: Trajectory image of November 8th, 2018 at 09:00:00, whose trajectory record duration is from 08:58:00 to 09:02:00

Figure 4.17.: Trajectory image of November 8th, 2018 at 15:00:00, whose trajectory record duration is from 14:58:00 to 15:02:00

Figure 4.18.: Trajectory image of November 8th, 2018 at 21:00:00, whose trajectory record duration is from 20:58:00 to 21:02:00

The image at 03:00:00 has much sparser lines than that of images at 09:00:00, 15:00:00, and 21:00:00, because only a few routes happen at early morning hours like 03:00:00.

Some straight small shallow lines randomly emerge in each example image. The lines are generated by flash moving of the coordinates, which may be the error of positional sensors. We only regard the lines as noise and expect that the computer vision neural network can ignore them.

4.2.4. Data Fusion for Multispectral Images

4.2.4.1 RGB 3 Phases Images Fusion Description

Data fusion is an effective way of synergistically combining information from various sources to better understand a given scene[14]. After the completion of Geographical Image, Roadmap Images and Trajectory Images, we will classify the 3 kinds of images according to the same time, and fuse the 3 phases using the image fusion framework.

Since one image consist of 3 phases: Red(R), Green(G), and Blue(B), each phase can be replaced by a grayscale of Geographical, Roadmap, and Trajectory Image individually. Where grayscale is a 1 phase black and white representation of an RGB 3 phases image. After the replacement is finished, the 3 new phases will be aligned together to generate the RGB 3 Phases Images.

4.2.4.2 Generation of Full RGB 3 phases Images

The process of Full RGB 3 phases image generation is:

A. Resize the Geographical Image from 1042*1043 (Chapter 4.2.1.3) to 896 pixels in width and 768 pixels in height.

B. Transform all Geographical, Roadmap and Trajectory Images(Size of (896,768,3)) to grayscale(Size of (896,768,1)).

C. Classify the three kinds of images according to the same time.

D. Fuse the 3 phases using the image fusion framework.

4.2.4.3 Full RGB 3 phases Images Demonstration

We generate 3603 Full RGB 3 phases Images from 00:00:00, October, 8th 2018 to 00:20:00, November 3rd, 2018, about 25 days. 144 images in each day with 10 minutes time interval, and additional three images for buffer.

Figure 4.19.: Mechanism of 3 phase image fusion framework

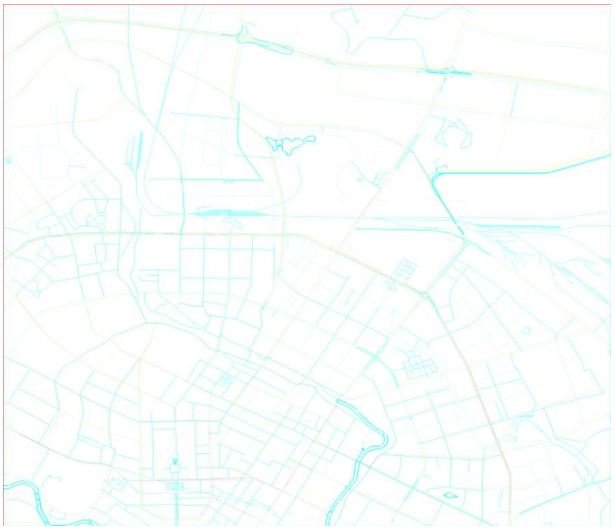


Figure 4.20.: Full 3 RGB image of November 8th, 2018 at 03:00:00

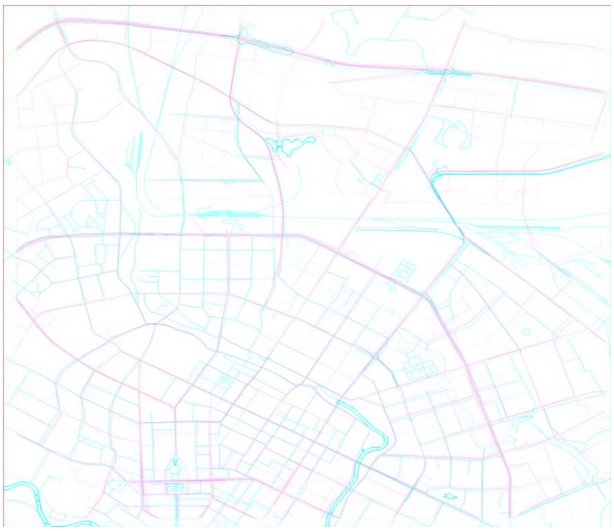


Figure 4.21.: Full 3 RGB image of November 8th, 2018 at 09:00:00

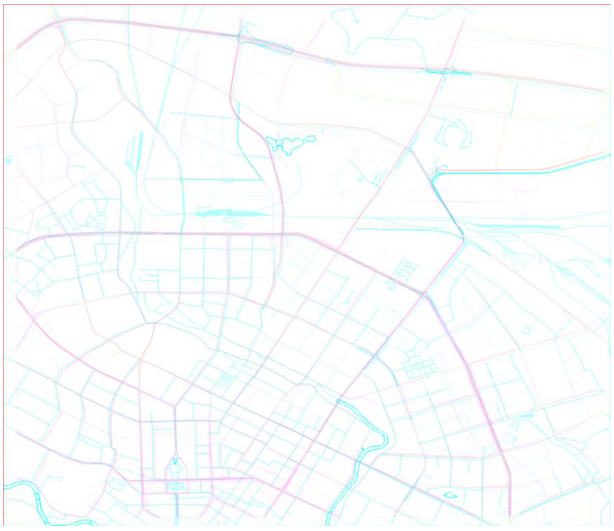


Figure 4.22.: Full 3 RGB image of November 8th, 2018 at 15:00:00

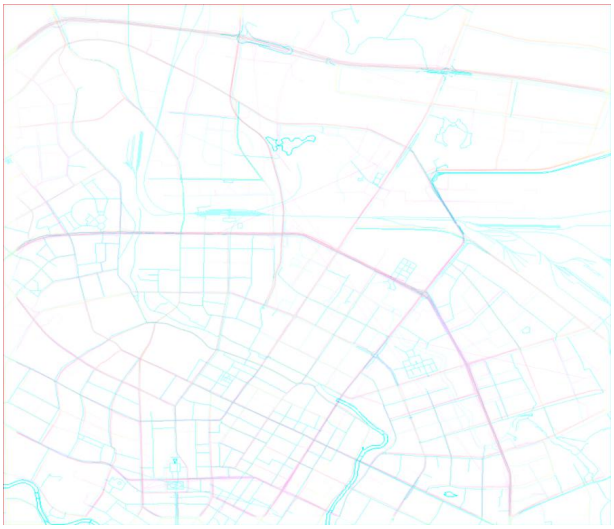


Figure 4.23.: Full 3 RGB image of November 8th, 2018 at 21:00:00

Since both the Roadmap Images and Trajectory Images are derived from Didi GAIA Dataset, the combinations of the two sets of images above are nearly perfect. However, due to unavoidable operation error, there is little deviation from Geographical Images as depicted.

4.2.5. Travel Time Index(TTI)

4.2.5.1 Output Values of TTI Description

TTI will be regarded as the traffic forecast value, based on Full RGB 3 phases images generated above. In deep learning, Full RGB 3 phases images will be the inputs of neural networks, and TTI after 10 minutes at the represented time of the corresponding image will be the outputs of neural networks. We will forecast both the individual road TTI and areal TTI.

4.2.5.2 TTI for Roads

TTI of 10 randomly selected individual roads will be forecast as examples, and the types of roads contain both expressways and urban roads. Most parts of the 10 individual roads are in the given research area.

4.2.5.3 TTI for a Specific Area

Besides TTI forecasting for individual roads, weight average TTI for a collection of roads inside a given area can also be deduced and forecasted.

Road ID	Name of Road	Color
281870	East 2nd Ring Road: Guojiaqiao West Street, Taoqi Road	Darkgray
281874	North 1st Ring Road: Jianshe North Road, Huapaifang Street	Blue
281876	East 1st Ring Road: Hongwasi Street, Jianshe North Road	Green
281885	Jianshe North Road: Fuqing Road, East Zhonghuan Road	Yellow
281896	Hongzhaobi Street: Renmin South Road, Shangnan Street	Black
282014	Da An West Road: Renmin Middle Road, North Street	Brown
282228	Dongtongshun Street: Taisheng North Road, Lion Lane	Red
282242	Xinghui East Road: Ma An South Road, Fuqing Road	Darkviolet
282252	Sanyou Road: First Ring Road, Second Ring Road	Deeppink
282272	Jiuliti Middle Road: Qunxing Road, Second Ring Road	Orange

Table 4.1.: Detailed information of 10 individual roads in the examples

Figure 4.24.: Position and shape of roads in Chinese Gaode Map

Assign Output TTI Prediction Area

As discussed in Chapter 3.2.3, transportation in Chengdu city is concentrated in urban. So if an overview of full transportation in the metropolis is needed, it is necessary to investigate transportation on the urban area. Since the layout of expressways in Chengdu City is a series of concentric circles, we assign a rectangular area that fully cover the Third Ring Avenue with the boundary of:

- Longitude Upper Bound: 104.162492
- Longitude Lower Bound: 103.983681
- Latitude Upper Bound: 30.726112
- Latitude Lower Bound: 30.594449

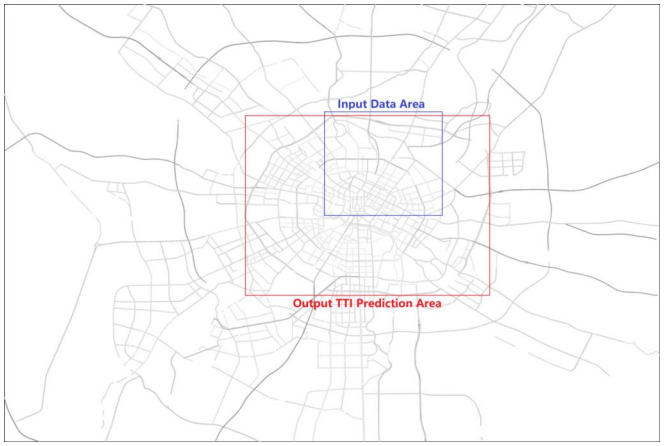


Figure 4.25.: Comparison of input data area and output TTI prediction area

Calculate the weight TTI

The steps of calculation of the weight of roads are:

A. Extract all the Well-Known Text(WKT) Geometry segments of roads in the TTI prediction area.

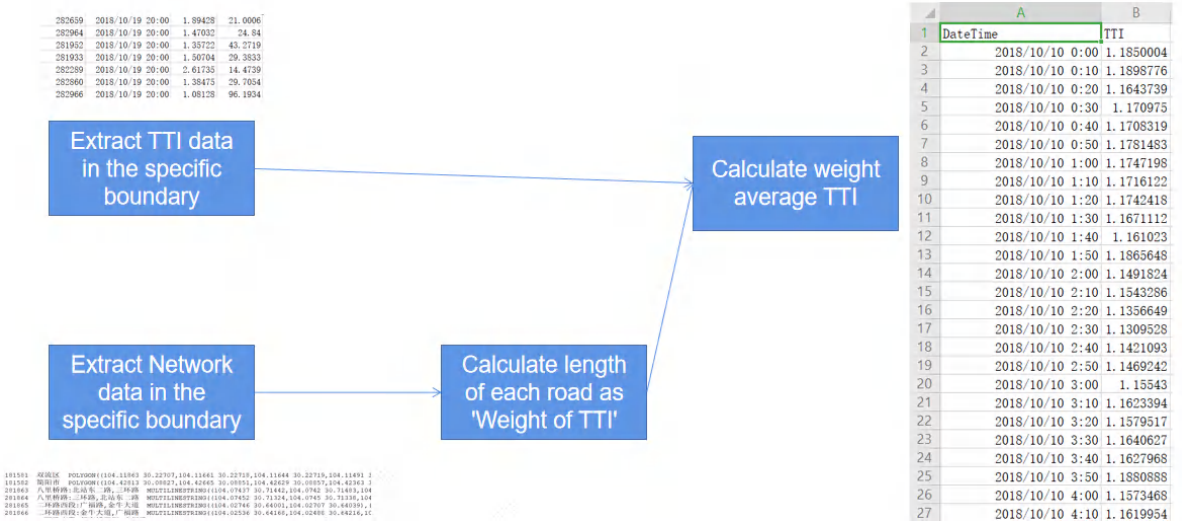
B. Calculate the length of segments based on Well-Known Text(WKT) Geometry in each road as 'weight of TTI'.

C. Extract TTI data from TTI and Average Speed dataset and group it by the same time. For example, group all TTI data from 1859 roads at 2018-10-10 00:00:00 together.

D. Based on 'weight of TTI', calculate the weighted average of one Unix time.

E. Do the previous grouping and calculating for every Unit time with 10min interval.

At last, we get the list of weight average TTI in the specific area at every timestamp.



	A	B
1	DateTime	TTI
2	2018/10/10 0:00	1.1850004
3	2018/10/10 0:10	1.1898776
4	2018/10/10 0:20	1.1643739
5	2018/10/10 0:30	1.170975
6	2018/10/10 0:40	1.1708319
7	2018/10/10 0:50	1.1781483
8	2018/10/10 1:00	1.1747198
9	2018/10/10 1:10	1.1716122
10	2018/10/10 1:20	1.1742418
11	2018/10/10 1:30	1.1671112
12	2018/10/10 1:40	1.161023
13	2018/10/10 1:50	1.1865648
14	2018/10/10 2:00	1.1491824
15	2018/10/10 2:10	1.1543286
16	2018/10/10 2:20	1.1356649
17	2018/10/10 2:30	1.1309528
18	2018/10/10 2:40	1.1421093
19	2018/10/10 2:50	1.1469242
20	2018/10/10 3:00	1.15543
21	2018/10/10 3:10	1.1623394
22	2018/10/10 3:20	1.1579517
23	2018/10/10 3:30	1.1640627
24	2018/10/10 3:40	1.1627968
25	2018/10/10 3:50	1.1880888
26	2018/10/10 4:00	1.1573468
27	2018/10/10 4:10	1.1619954

Figure 4.26.: Flow chart of weight average TTI generation

Mathematical Approximation

Assuming that in an area A, the number of roads in the area A is p, then the mathematical representation can be:

$$A = \{s_1, s_2, s_3, ..., s_p\}$$

While A is a set, and the size of set A is p, s_q is the representation of individual roads where $1 \geq q \leq p$

Based on the process of areal weighted TTI calculation above, the calculation formula of areal weighted TTI is:

$$TTI_{weight} = \frac{\sum_{i=1}^{N_1} L_i \cdot TTI_i + \sum_{j=1}^{N_2} L_j \cdot TTI_j + ... + \sum_{\lambda=1}^{N_p} L_\lambda \cdot TTI_\lambda}{\sum_{i=1}^{N_1} L_i + \sum_{j=1}^{N_2} L_j + ... + \sum_{\lambda=1}^{N_p} L_\lambda} \tag{4.5}$$

Among the numerator, the individual TTI can be represented by:

$$TTI = \frac{\sum_{i=1}^{N} \frac{L_i}{V_i} \cdot W_i}{\sum_{i=1}^{N} \frac{L_i}{V_{free_i}} \cdot W_i} \tag{4.6}$$

which has been indicated in in chapter 2.1.4, In a collection of links s.

$$s = \{Link_1, Link_2, ..., Link_N\}$$

Where size of s is N, length of link is L_i, weight of link is W_i, free flow speed is V_{free_i}, and the realtime speed is V_i.

However, a link is a part of a road, and a road is a part of the area. Mathematically:

$$L \in s, s \in A$$

By the definition of TTI, the formula of areal weighted TTI can be derived from the road TTI equation in (4.6) by expanding both the numerator and denominator:

$$TTI_{weight'} = \frac{\sum_{i=1}^{N_1} \frac{L_i}{V_i} \cdot W_i + \sum_{j=1}^{N_2} \frac{L_j}{V_j} \cdot W_j + ... + \sum_{\lambda=1}^{N_p} \frac{L_\lambda}{V_\lambda} \cdot W_\lambda}{\sum_{i=1}^{N_1} \frac{L_i}{V_{free_i}} \cdot W_i + \sum_{j=1}^{N_2} \frac{L_j}{V_{free_j}} \cdot W_j + ... + \sum_{\lambda=1}^{N_p} \frac{L_\lambda}{V_{free_\lambda}} \cdot W_\lambda} \tag{4.7}$$

The formulas of areal weighted TTI derived by the two methods are different. To eliminate the difference, we need to prove:

$$TTI_{weight} = TTI_{weight'} \tag{4.8}$$

Under some appropriate assumptions which has been indicated in chapter 4.1.

Here two assumptions are made:

Assumption 1: $W_i = 1$ for all i.

Because in macroscopic research, we have omitted the width of the road, so weights of roads will not be considered. Thus only the lengths of roads will be considered.

Assumption 2: $V_{free_i} = V_{free_j} = V_{free}$ for all i and j.

Because in urban road, policy makers have an identical maximum speed to regulate all the urban road vehicles.

So:

$$
\begin{aligned}
TTI_{weight'} &= \frac{\sum_{i=1}^{N_1} \frac{L_i}{V_i} \cdot W_i + \sum_{j=1}^{N_2} \frac{L_j}{V_j} \cdot W_j + ... + \sum_{\lambda=1}^{N_p} \frac{L_\lambda}{V_\lambda} \cdot W_\lambda}{\sum_{i=1}^{N_1} \frac{L_i}{V_{free_i}} \cdot W_i + \sum_{j=1}^{N_2} \frac{L_j}{V_{free_j}} \cdot W_j + ... + \sum_{\lambda=1}^{N_p} \frac{L_\lambda}{V_{free_\lambda}} \cdot W_\lambda} \\
&= \frac{\sum_{i=1}^{N_1} \frac{L_i}{V_i} \cdot V_{free} + \sum_{j=1}^{N_2} \frac{L_j}{V_j} \cdot V_{free} + ... + \sum_{\lambda=1}^{N_p} \frac{L_\lambda}{V_\lambda} \cdot V_{free}}{\sum_{i=1}^{N_1} L_i + \sum_{j=1}^{N_2} L_j + ... + \sum_{\lambda=1}^{N_p} L_\lambda} \\
&= \frac{\sum_{i=1}^{N_1} \frac{L_i/V_i}{1/V_{free}} + \sum_{j=1}^{N_2} \frac{L_j/V_j}{1/V_{free}} + ... + \sum_{\lambda=1}^{N_p} \frac{L_\lambda/V_\lambda}{1/V_{free}}}{\sum_{i=1}^{N_1} L_i + \sum_{j=1}^{N_2} L_j + ... + \sum_{\lambda=1}^{N_p} L_\lambda} \\
&= \frac{\sum_{i=1}^{N_1} L_i \cdot \frac{L_i/V_i \cdot W_i}{L_i/V_{free} \cdot W_i} + \sum_{j=1}^{N_2} L_j \cdot \frac{L_j/V_j \cdot W_j}{L_j/V_{free} \cdot W_j} + ... + \sum_{\lambda=1}^{N_p} \cdot L_\lambda \frac{L_\lambda/V_\lambda \cdot W_\lambda}{L_\lambda/V_{free} \cdot W_\lambda}}{\sum_{i=1}^{N_1} L_i + \sum_{j=1}^{N_2} L_j + ... + \sum_{\lambda=1}^{N_p} L_\lambda} \\
&= \frac{\sum_{i=1}^{N_1} L_i \cdot TTI_i + \sum_{j=1}^{N_2} L_j \cdot TTI_j + ... + \sum_{\lambda=1}^{N_p} L_\lambda \cdot TTI_\lambda}{\sum_{i=1}^{N_1} L_i + \sum_{j=1}^{N_2} L_j + ... + \sum_{\lambda=1}^{N_p} L_\lambda} \\
&= TTI_{weight}
\end{aligned}
\tag{4.9}
$$

4.3. Deep Learning

In chapter 4.2.5, we have specified that Full RGB 3 phases images will be the inputs of neural networks, and TTI after 10 minutes at the represented time of the corresponding image will be the outputs of neural networks.

In the deep learning part, we will use computer vision neural networks to do the TTI prediction.

4.3.1. Basic Settings

4.3.1.1 Computation Platforms

Graphics Processing Unit (GPU) is essential for deep learning. There are two computation platforms equipped with GPU to use: one is from Kaggle Kernel[36], the other is from the High-performance computing platform of Shanghai Jiao Tong University Student Innovation Center. The hardware settings of these two environments are:

Kaggle

•Name of GPU: Nvidia Tesla-P100.
•Internal Storage of GPU: 16280MB.
•Maximum allowed time for usage: 9 hours every time, and 36 hours every week.

High-performance computing platform

•Name of GPU: Nvidia GeForce GTX 1080 Ti.
•Internal Storage of GPU: 11178MB.
•Maximum allowed usage time: no restriction.

4.3.1.2 Deep Learning Framework

Torch is a scientific computing framework with wide support for machine learning algorithms that puts GPUs first[37]. To construct the torch framework, we use python to construct PyTorch framework for deep learning[38].

Pytorch can provide two high-level features:

•Tensor computation (like NumPy) with strong GPU acceleration.
•Deep neural networks built on a tape-based autograd system[38].

4.3.2. Dataset Splitting and K-Fold Cross-Validation

First, we split the datasets into a training dataset, validation dataset, and test dataset[39].

A test dataset is a set of examples used only to assess the performance of the deep learning model. In our task, the test dataset consists of 144 pairs of the input images and output TTI, where the time of images is from 00:10:00 November 1st, 2018 to 00:10:00 November 2nd, 2018.

A training dataset is a dataset of examples used during the learning process and is used to fit the parameters of the deep learning model. A validation dataset is a dataset of examples used to tune the hyperparameters of the deep learning model.

We use K-fold cross-validation to split the training dataset and validation dataset. After the exclusion of the test dataset, 24 days of datasets are available. Considering both the computation cost and data utilization, we use 6 fold cross-validation. Thus, 20 days of data for the training dataset, 4 days of data for the validation dataset, and the training for 6 individual folds.

Fold	Training Dataset	Validation Dataset
1	Oct. 8th - Oct. 27th	Oct. 27th - Nov. 1st
2	Oct. 8th - Oct. 23rd Oct.27th - Nov. 1st	Oct. 23rd - Nov. 27nd
3	Oct. 8th - Oct. 19th Oct.23rd - Nov. 1st	Oct. 19th - Oct. 23rd
4	Oct. 8th - Oct. 15th Oct.19th - Nov. 1st	Oct. 15th - Oct. 19th
5	Oct. 8th - Oct. 11th Oct.15th - Nov. 1st	Oct. 11th - Oct. 15th
6	Oct. 11th - Nov. 1st	Oct. 8th - Oct. 11th

Table 4.2.: 6 folds dataset split

4.3.3. Loss Function Choosing

The purpose of loss functions is to compute the quantity that a model should seek to minimize during training[40]. Briefly, an optimization problem seeks to minimize a loss function.

There are two classical loss functions to use for this task, L1 Loss and L2 Loss. Where L1 Loss stands for Least Absolute Deviations, and L2 Loss function stands for Least Square Errors.

$$L1LossFunction = \sum_{i=1}^{n} |y_{true} - y_{predicted}| \tag{4.10}$$

$$L2LossFunction = \sum_{i=1}^{n} (y_{true} - y_{predicted})^2 \tag{4.11}$$

From the data analysis in chapter 3.4.2, we figure out that TTI values outliers, which can less than 0.2 or bigger than 500, will largely affect the performance of the deep learning model if we use L2 Loss. Thus, we use L1 Loss as the loss function in backward propagation.

4.3.4. Learning Rate Reduction In Training

As the model gradually converges to the target, the learning rate should be reduced, which is indicated in Chapter 2.2.5. In PyTorch, function ReduceLROnPlateau is a learning rate scheduler that allows dynamic learning rate reducing based on some validation measurements[41]. This scheduler can read a metrics quantity (like validation loss) and if no improvement is seen for a 'patience' number of epochs, the learning rate will be reduced[41]. In our task, the initial learning rate is 0.001, and it will be reduced as a factor of 0.3 if the validation loss failed to decrease[41].

```
CLASS torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.1,
      patience=10, threshold=0.0001, threshold_mode='rel', cooldown=0, min_lr=0, eps=1e-08,  [SOURCE]
      verbose=False)
```

Figure 4.27.: Description of learning rate scheduler ReduceLROnPlateau

4.3.5. Historical Average Baseline

In Chapter 1.1.2, the historical average is one of the easily implemented but low accurate methods of traffic forecast, which can be used as a baseline of deep learning performance evaluation. When the historical average method is applied, one potential assumption is implied: The dataset is highly periodic so that the regular pattern of historical data and future data are similar. In chapter 3.4.2, an explanation of periodic features in traffic data has been made, so the historical average method may result in a good performance. However, the good performance highly depends on luck, because the simple historical data can not forecast any possible alternative situation in the test dataset.

We compute the average TTI values of the previous 24 days at the same time as the test dataset. For instance, the prediction TTI value at 12:00:00 on November 2nd is the average of all TTI values from October 8th to November 1st at 12:00:00.

4.3.6. Single LSTM Without Any Image Fusion Framework

As indicated in Chapter 2.2.2, single LSTM models perform well in some tasks of traffic forecast without any data fusion framework. In our TTI forecasting, one single LSTM model with two sets of inputs will be tested individually. One of the input data includes historical TTI values, another input data include variables of time, average speed, and length of the road.

4.3.6.1 Single LSTM With Single Value Input

In a python code example of passenger prediction, the author uses PyTorch LSTM to predict the total number of traveling passengers in a specific month by flight[42]. In the code example, the dataset consists of the number of monthly traveling passengers from 144 continuous months since January 1949, among which the first 132 values are training dataset, and the last 12 values are testing dataset. The corresponding LSTM model has input size = 12, hidden layer size = 100, and output size = 1, which means the model uses the 12 values from 12 continuous months to predict the following 1 value in 1 month.

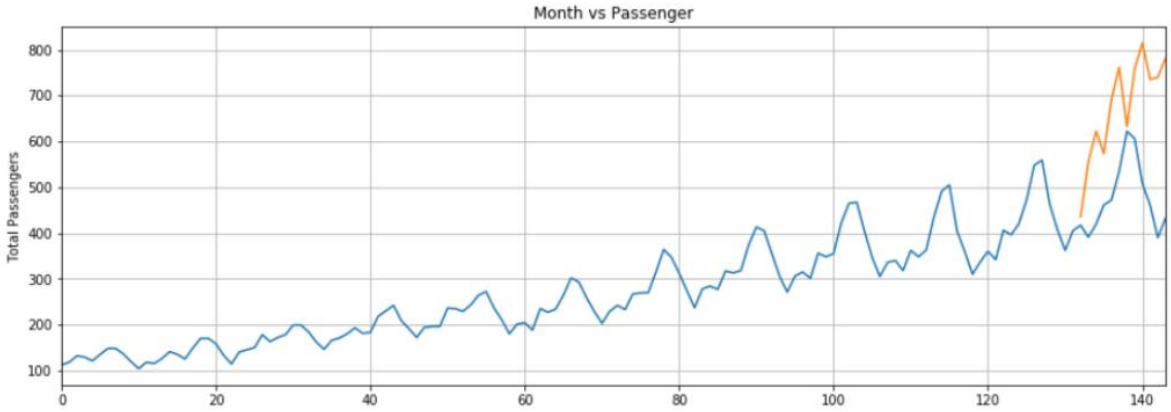


Figure 4.28.: Result of flight passenger prediction[42].

In our task, we will use a similar model to do the TTI prediction after 10 minutes. Since the size of our test dataset is 144, whose period is one day, the corresponding LSTM model has input size = 144, hidden layer size = 100, and output size = 1. We construct 11 LSTM models to predict the TTI values of 10 individual roads and 1 TTI value of area mentioned in chapter 4.2.5. Based on the structure of the LSTM model indicated below, when predicting the last TTI value (TTI value at 00:10:00 November 2nd, 2018) in the test dataset, the inputs should be the previous 144 TTI values from the last 24 hours.

However, among the 144 inputs, 143 of them are in the test dataset. As indicated in chapter 4.3.2, we have strictly divided the whole dataset into 3 disjoint subsets: training dataset, validation dataset, and test dataset. Thus we do not have an opportunity to use any part of the test dataset to do the prediction of the test dataset. In practice, if people already have the TTI values in the future, it is unnecessary to forecast the TTI values in the future.

Based on the analysis above, the only applicable prediction method of test dataset is:

1. Use 144 TTI values from 00:10:00 October 31st, 2018 to 00:10:00 November 1st, 2018 to predict the TTI value at 00:20:00 November 1st, 2018.

2. Take the prediction value at 00:20:00 on November 1st, 2018 as an input of the next prediction.

3. Use 144 TTI values from 00:20:00 October 31st, 2018 to 00:20:00 November 1st, 2018 to predict the TTI value at 00:20:00 November 1st, 2018.

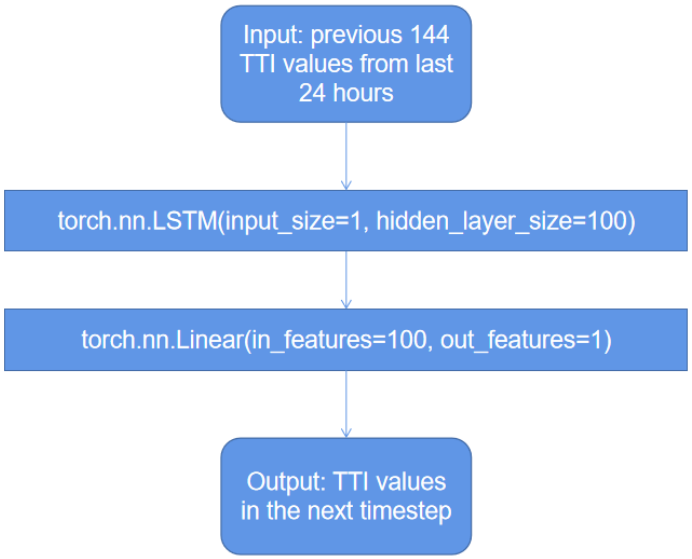


Figure 4.29.: Structure of the LSTM model with single value input

4. Repeat and forward the steps 1 to 3 for the rest 143 TTI values, and finish the test data prediction.

5. Compare the predicted TTI values and TTI values in the test dataset, and compute the loss.

4.3.6.2 Single LSTM with Multiple Values Inputs

One of the shortcomings of the model in chapter 4.3.6 is: using predicted values to do prediction will amplify the error between predicted values and actual values. To avoid this problem, we apply a single LSTM model with multiple values as inputs. In a master thesis, the author uses 10 different values as inputs to predict the travel time[43].

Similar to the inputs shown in the graph, in our TTI and average speed dataset, time information is also available when recording the TTI values. Besides the time information, the average speed is also available. In-Network Dataset, we can also compute the total length of selected road to acquire road length data.

However, the above data is all we can use, and none of the Routing Dataset can be used in a single LSTM model. As indicated in Chapter 3.3.1, there is no direct connection between Routing Dataset and the other two datasets. Since the source of Routing Dataset is different from the other two datasets, researchers will not pay attention to Routing Dataset if they do not use the graphical data fusion framework.

The only possible connection between Routing Dataset and the other two datasets

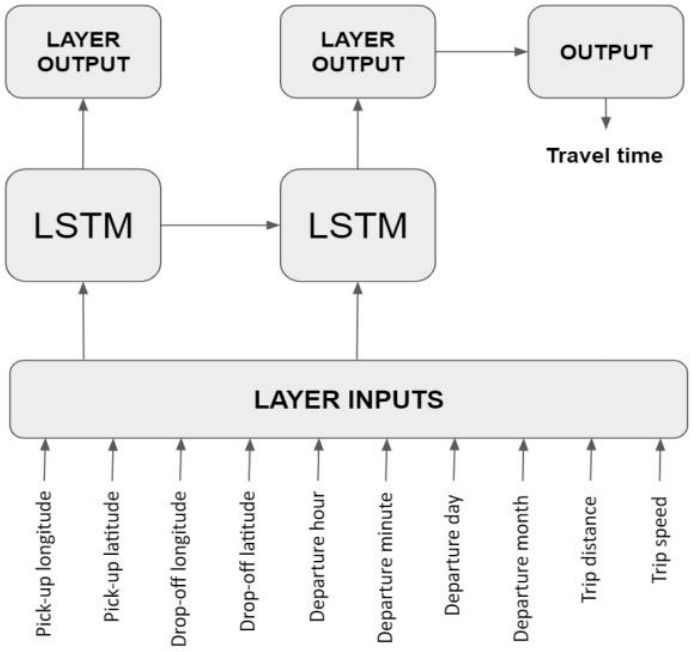


Figure 4.30.: LSTM Model structure used to predict travel time in a master thesis[43].

	A	B	C	D	E	F	G	H
1	Name	Month	Day	Hour	Minute	Speed	Length	TTI
2	281870	10	8	0	10	40.0494	0.0786951	1.19291
3	281870	10	8	0	20	39.5863	0.0786951	1.17097
4	281870	10	8	0	30	40.3265	0.0786951	1.14083
5	281870	10	8	0	40	41.3934	0.0786951	1.1413
6	281870	10	8	0	50	41.3765	0.0786951	1.12604
7	281870	10	8	1	0	41.8318	0.0786951	1.088
8	281870	10	8	1	10	43.3242	0.0786951	1.16171
9	281870	10	8	1	20	40.6256	0.0786951	1.10095
10	281870	10	8	1	30	42.893	0.0786951	1.11411
11	281870	10	8	1	40	42.3639	0.0786951	1.14889
12	281870	10	8	1	50	40.982	0.0786951	1.10153
13	281870	10	8	2	0	42.8534	0.0786951	1.16892
14	281870	10	8	2	10	40.3874	0.0786951	1.1032
15	281870	10	8	2	20	42.774	0.0786951	1.13432
16	281870	10	8	2	30	41.6107	0.0786951	1.11172
17	281870	10	8	2	40	42.5189	0.0786951	1.16128
18	281870	10	8	2	50	40.6023	0.0786951	1.08969
19	281870	10	8	3	0	43.0444	0.0786951	1.19095

Figure 4.31.: Demonstration of a part of multiple inputs in Microsoft Excel.

is the positional coordinates: longitude and latitude. Thus, without the graphical data fusion framework, the only method to reveal the geographical connection between the different datasets is analytic geometry. However, analytic geometry has the following fatal shortcomings in revealing the connection between the different datasets:

•Extreme complexity of computation: Since the shape of the road is described by WKT geometry in chapter 2.1.2.2, which only gives the endpoints of links on roads. To detect whether the positions of vehicles in the Routing Dataset is on specific roads, we need to calculate the distance between each position and each line, from over 24 GB Routing Dataset.

•High uncertainty in positional coordinating: After successfully calculate the distance between each position and each line, we need to decide the threshold of distance for deciding whether the positions of vehicles in Routing Dataset is on specific roads. However, because of both possible positional error in GPS and vast two-way road layouts in urban roads, it is extremely difficult to confirm that a vehicle is on a specific road.

In this task, the corresponding LSTM model has input size = 7, hidden layer size = 100 and output size = 1. The input values are ID of road, month, day, hour, minutes, and average speeds when recorded the TTI value, and the length of the road. The output values are TTI values after 10 minutes. We construct 11 LSTM models to predict the TTI values of 10 individual roads and 1 TTI value of area mentioned in chapter 4.2.5.

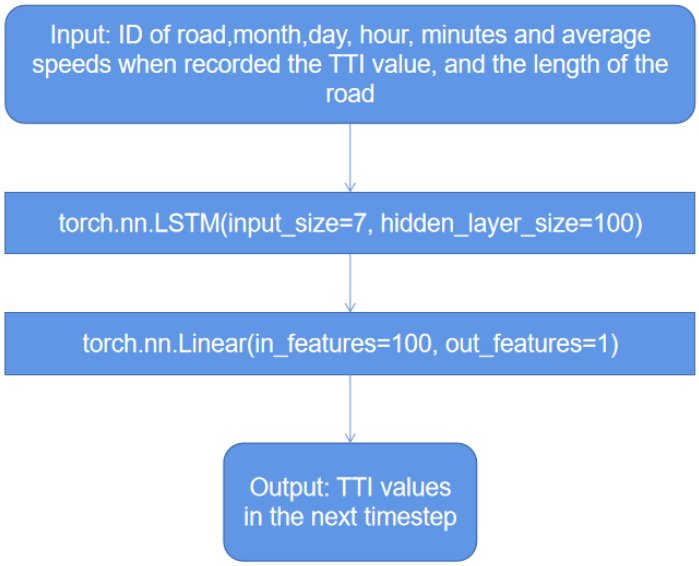


Figure 4.32.: Structure of the LSTM model with multiple values input

4.3.7. Resnet With Image Fusion Framework

There are five types of Resnets with different layers: 18, 34, 50, 101, and 152.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 4.33.: Structures of Resnets with different layers: 18, 34, 50, 101, and 152 [18].

Considering both the computation cost and model accuracy, we use Resnet34 as the backbone of our neural network. To connect the number of output and that of the original output from Resnet34, we use two fully connected layers with the number of features: 512, 4096, and 11, to replace the last fully connected layer of the original Resnet34. 512 is the number of output features in the Resnet34 backbone. 4096 is the number of features in the middle of two fully connected layers. 11 is the number of TTI values to be predicted, which is 10 TTI values of 10 individual roads, and 1 TTI value of the TTI prediction area indicated in Chapter 4.2.5.

In terms of data loading of the dataset, we load the Full RGB 3 phases image from 00:10:00 October 8th, 2018 to 00:10:00 November 2nd, 2018 as inputs, and the TTI values 10 minutes later correspondingly, that is, from 00:20:00 October 8th, 2018 to 00:20:00 November 2nd, 2018 as outputs. The period of the dataset is 25 days.

For the detailed information of how we construct the networks, how we load the data into them, and how we do predicitons, please refer to the codes in appendix part.

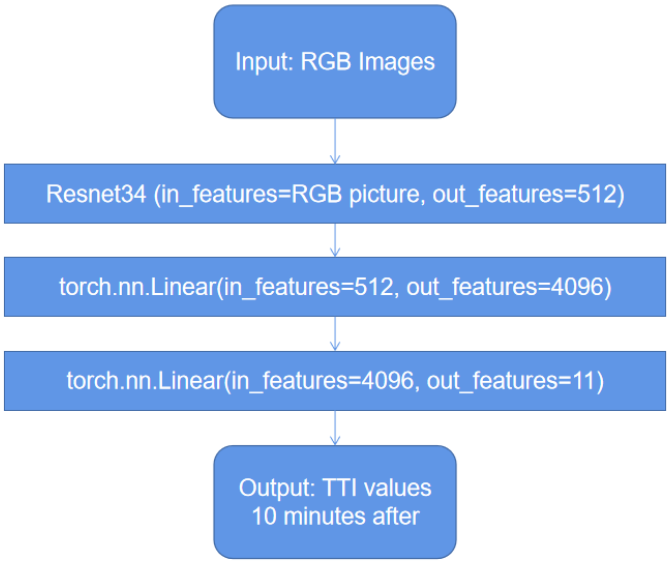


Figure 4.34.: Structure of neural network with Resnet34 backbone and two fully connected layers.

5. Result

To make the deep learning models converge, models should be trained for a sufficient number of epochs. The criteria of deciding the number of epochs will be: when the learning rate is reduced to 10^{-6} or a smaller number, the training will be terminated, as is indicated in chapter 2.2.4.

Based on the criteria, for LSTM models for both single input and multiple inputs, the models will converge after about 30 epochs. Comparing to LSTM models, the Resnet34 model has a much more complicated structure, thus the Resnet34 model will converge after about 90 epochs.

Although the method of using test dataset to predict the test dataset has been denied in chapter 4.3.6.1, the performance is still shown with the column name 'LSTM' below:

5.1. Test Loss Comparison of Different Models

Road ID	Historical Average	Resnet34	LSTM (Single Input)	LSTM (Multiple Input)	LSTM
281870	0.13751	0.11284	0.23952	0.69540	0.02241
281874	0.07473	0.06299	0.22158	0.58185	0.04958
281876	0.08684	0.04913	0.18057	0.48999	0.03003
281885	0.06096	0.06370	0.37082	0.60964	0.03108
281896	0.10174	0.09482	0.28598	0.61231	0.24025
282014	0.06234	0.05960	0.40116	0.53485	0.11554
282228	0.17517	0.17066	0.51937	0.80210	0.42007
282242	0.11649	0.11439	0.48036	0.72747	0.14938
282252	0.10599	0.10186	0.15552	0.75103	0.06292
282272	0.19019	0.19143	0.38607	0.73633	0.05428
Area	0.04659	0.03400	0.16136	0.25482	0.01586
Total	0.10532	0.09595	0.30930	0.61780	0.1083

Table 5.1.: Average Loss Comparison of Different Models

5.2. Graphical Demonstration of Results

By checking the values of total test losses of different models, we figure out that two LSTM models with both Single Input and Multiple Input are failed to exceed the baseline. So we will only demonstrate the performance of the rest of the models. Here only the plot of areal weighted TTI is shown, the plots of individual roads can be found in the appendix.

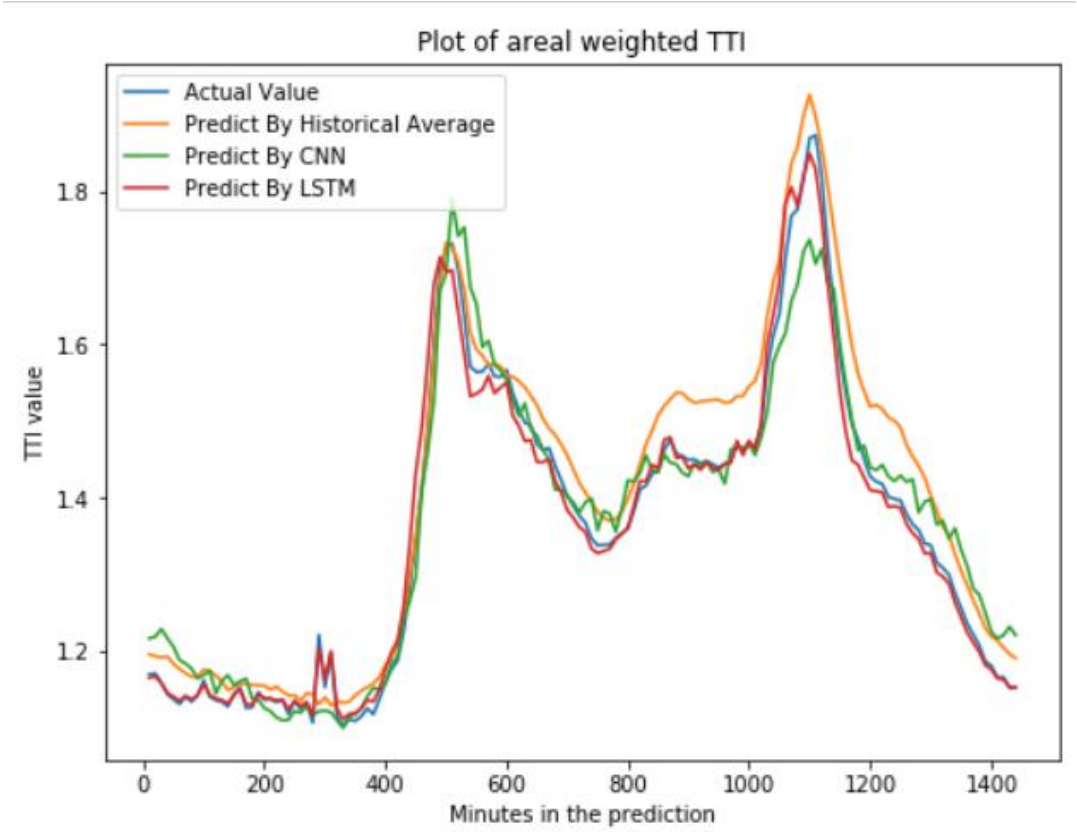


Figure 5.1.: Performance of models in an area

5.3. Demonstration of Learning Procedure of Deep Learning Models

To visualize a part of the training process, the graphs for prediction results of Resnet34 in areal TTI at 30 epochs, 60 epochs, and 90 epochs will be shown.

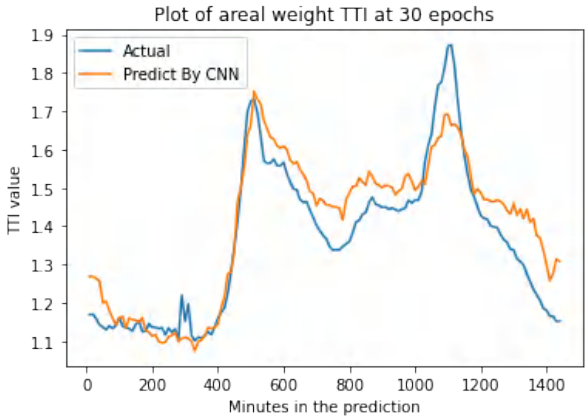


Figure 5.2.: Prediction Result of Resnet34 in areal TTI at 30 epochs

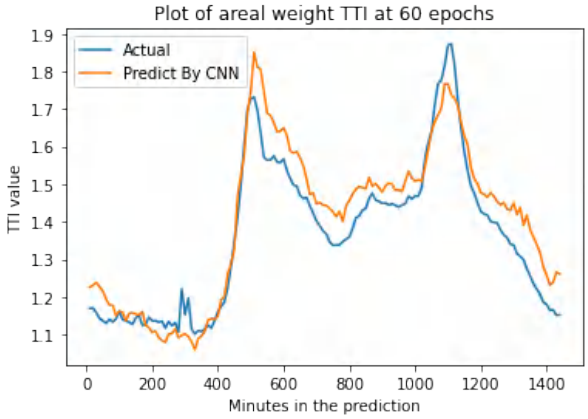


Figure 5.3.: Prediction Result of Resnet34 in areal TTI at 60 epochs

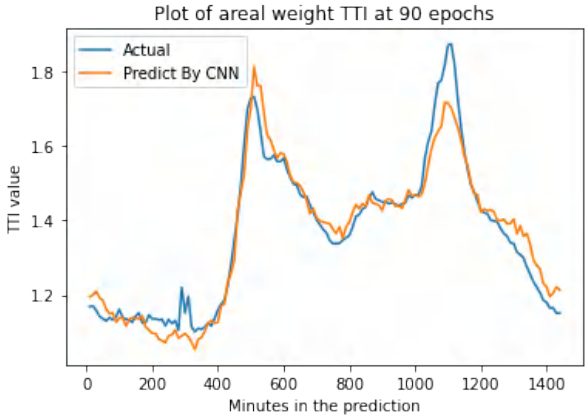


Figure 5.4.: Prediction Result of Resnet34 in areal TTI at 90 epochs

6. Conclusion and Discussion

6.1. Conclusion

6.1.1. Overall Conclusion

Set the loss of historical average model as the baseline, the performance of other models in terms of total average loss are:

Resnet34: 0.09595/0.10532 = 0.911 = 91.1%.

LSTM model with single input: 0.3093/0.10532 = 2.937 = 293.7%

LSTM model with multiple input: 0.61780/0.10532 = 5.866 = 586.6%

LSTM model use test dataset to do prediction: 0.1083/0.10532 = 1.028 = 102.8%

Among the models indicated above, only the performance of Resnet with images fusion framework exceeds the baseline by about 9%. The performance of LSTM with single input fall behind the baseline by about 190%, the performance of LSTM with multiple inputs fall behind the baseline by about 586.6%, the performance of LSTM use test dataset to do prediction behind the baseline by about 2.8%

An interesting observation for the result is that: for the LSTM model to use a test dataset to do prediction, the variance of performance in individual roads is large. For example, on some roads like No. 281870 and No. 281885, the model has the best performances. But on some roads like No. 282228 and No. 281896, the model has bad performances.

The reason for this is partly explained in chapter 4.3.5: the single input of historical TTI will make the performance of models highly dependent on periodic historical experience. If the prediction data is similar to the historical data, the performance of models will be excellent. However, any possible alternative situations will largely spoil the performance.

Besides, as indicates in chapter 2.2.2, LSTM models are much more capable to learn historical status than the historical average. In other words, LSTM models are much more dependent on historical status than the historical average. Thus, when the prediction data is similar to the historical data, LSTM models have the advantages to reduce the difference between historical data and prediction data. However, when the prediction data is different from the historical data, LSTM models have disadvantages to adapt the features of new alternatives in prediction data. Although we made a

trial to input different kinds of data into single LSTM models, the structure of LSTM will be damaged because of the independence of the different kinds of inputs.

Comparing to single LSTM models that will no feature engineering, the Resnet34 model with image fusion framework not only has an overall good performance, but the variance of the performance between different roads is low as well. Thus the robustness of the Resnet34 model is much more stable than that of single LSTM models after different feature engineering of the dataset.

6.1.2. Failure of Single LSTM Models

6.1.2.1 Underfitting During Prediction

As indicated in chapter 4.3.6.1, using predicted values to do prediction will amplify the error between predicted values and actual values. By looking at the graph depicting the performance of LSTM models, we figure out that the prediction line is much smoother than the actual line. Thus the error is not amplified. However, the smooth line indicates the underfitting of the test data, where underfitting is mainly because of the lack of sufficient data. Thus the failure of single LSTM models with historical Travel Time Index inputs can be interpreted as follows: the prediction of test data by models can be regarded as using a function $f(x)$ to compute the prediction values y by using inputs x. Take the length of inputs equals 144 as an example, if we can use the test data to predict the test data, the length of inputs is 288 (Travel Time Indexes from 00:10:00 October 31st, 2018 to 00:10:00 November 2nd, 2018). However, if we cannot use the test data to predict the test data, we need to use the prediction values to predict the test data, among which the prediction values are also generated by the previous 144 inputs (Travel Time Index from 00:10:00 October 31st, 2018 to 00:10:00 November 1st, 2018). In conclusion, the number of inputs to predict test data reduce 50%, which results in the underfitting of the test data.

6.1.2.2 Fragmentary Discrete Data

As indicated in chapter 2.2.2, LSTM models are suitable for the prediction of continuous action. However, the 10min time interval of TTI recordings breaks the continuous data into fragmentary discrete data.

Take the graph of discrete average speeds of road No. 283509 as an example, at point A, the average speed on the road is only about 20 km/h, contrast shapely to the points nearby. Thus the value of speed at point A can be regarded as an outlier. However, in the actual situation, the speed at point A is likely to be a normal value in a congestion time. Maybe the value of speeds at 10 seconds before and 10 seconds after point A are close to 20km/h, which can continuously connect point

A because the gradual acceleration and deceleration is essential for the changes of average speed.

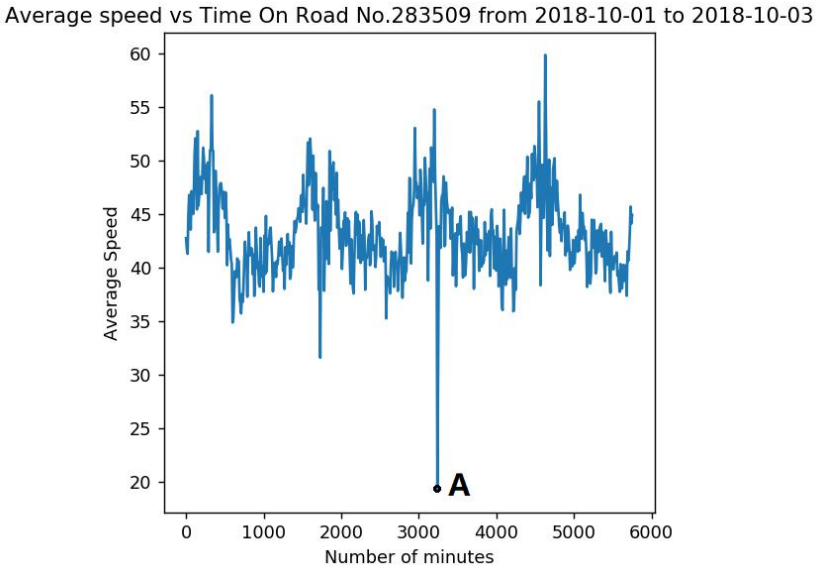


Figure 6.1.: Average speeds of road No.283509 with an outlier point A

In conclusion, point A is not an outlier but a victim of low sampling frequency. Since the time interval of data sampling is too long (10 minutes), the original continuous speed data is broken into fragmentary discrete speed data, whose values of speed at adjacent points are nearly irrelevant. When the LSTM model tries to fit the data, the fragmentary discrete data will confuse and spoil the model.

6.1.2.3 Lack of Spatial Features Input

Except for the image fusion framework, there is no way to input spatial features into the LSTM. However, in chapter 6.1.1, we have figured out that the dataset is heterogeneous concerning the spatial features. In other words, in the datasets used in this research, the difference of features in individual roads hugely affects the results. Besides, in chapter 3.1, the structure of our datasets shows that the proportion of spatial data is much larger than the proportion of time-related data. Thus, although the performance of LSTM models is brilliant in some of the traffic forecast tasks, as travel time prediction, the performance of LSTM models is bad in our Travel Time Indexes prediction because of the specific structure of our datasets.

As we discussed in chapter 1.3, checking the applicability of deep learning models based on the structure of datasets before the application of deep learning models should not be neglected.

6.1.3. Advantages of Image Fusion Framework

6.1.3.1 Dataset Augmentation

The image fusion framework can largely augment the original dataset. Without the framework, researchers are likely to only focus on the original dataset, which only contains limited information. After the perception of the framework, researchers can actively search the possible multi-sources datasets related to the original datasets under the guidance of the framework. With the merging of datasets derived from multi-sources, the performance of models can be improved by dataset augmentation.

6.1.3.2 Merging Continuous Data and Discrete Data

The image fusion framework can merge continuous data and discrete data. As we discussed in chapter 6.1.2, the discontinuous Travel Time Index and Average Speed Dataset degrade the performance of the LSTM models. In the framework, the continuous Routing Dataset can be merged with the discontinuous Travel Time Index and Average Speed Dataset. During the merging process, the duration of the recording time is decided as an intermediate value between the maximum (5 minutes) and the minimum (1 second), which not only preserve the continuous features of the dataset but also correspond to the discrete features of images in another phase.

6.1.3.3 Spatial Feature Inputs

The image fusion framework can visualize the spatial features of the datasets. As we discussed in chapter 4.3.6, without the graphical data fusion framework, it is impossible to connect the spatial relationships between different datasets. With the image fusion framework, the spatial features from different datasets can be merged, and their relationship will be implied in the generated RGB 3 phases images.

6.2. Discussion

6.2.1. Shortcomings of Image Fusion Framework

6.2.1.1 Omit the feature engineering of outliers

In most of the datasets, the number of outliers is much smaller than the number of normal values. Although the TTI outliers have been detected in Chapter 3, we choose to ignore TTI outliers. In Chapter 4.3.6, the L1 Loss function is used to minimize the influence of outliers, which makes the model difficult to predict the outliers.

6.2.1.2 Low Precision of Individual Roads

In Chapter 4.1, to reduce the computation and time consumption, we choose to reduce the precision from 0.00001 degrees of longitude and latitude to 0.0001 degrees, and we also ignore the width of roads. Although macroscopic traffic forecast of areal TTI value will not be affected significantly, the approximation will have a negative influence on microscopic traffic forecast of TTI in individual roads.

6.2.1.3 Unbalanced Weight of 3 Phases

When images generated from different sources are fused into individual phases separately, we potentially assume that the 3 phases are equal-weighted. However, among all the datasets, Trajectory Data has the highest number of characteristics of them, and Geographical Data has the least number of characteristics among them. The ignorance of weight configuration in the data fusion framework will fail to utilize sufficient information from Trajectory Data.

6.2.1.4 Bad Feature Engineering of Speed

During the Feature Engineering of speed, there are some randomly shallow lines emerge in Chapter 4.2.3. We just ignore them because the lines are treated as noise. However, if we can analyze the lines, we may find some insights into the speeds during the feature engineering of Trajectory Data. In the full image depicted in Chapter 4.2.4, the blue and red lines are clear but the green lines are vague, which means the features of Roadmap Data are not depicted in the full image.

6.2.2. Possible Improvement Ways of Shortcomings

6.2.2.1 Search the Possibility of TTI Outliers

The emergence of TTI outliers is mainly because of the rapid reduction of vehicle speeds, but the transition status of deceleration is still existed and can be observed in a continuous recording of traffic status.

Trajectory Data has the highest number of characteristics among the raw data, and the time interval of trajectory is only several seconds. Thus the Trajectory Data has plenty of detailed continuous features of traffic which can forecast the possibility of the emergence of TTI outliers.

6.2.2.2 Increase the Precision by Image Segmentation

In one of Citywide Crowd Flows Prediction, researchers partition a city into an $I * J$ grid map based on the longitude and latitude where a grid denotes a region[33]. Under a given computation power, split one large resolution image into several pieces of small resolution images can avoid the precision reduction.

6.2.2.3 Assign Suitable Weights for Each Phase

In the Citywide Crowd Flows Prediction indicated in Chapter 6.2.2, researchers also fuse the outputs of the first three components based on parameter matrices, which assign different weights to the results of different components in different regions[33]. If we introduce an attention mechanism into the framework to adjust the weight of each phase, the appropriate utilization of features from different phases can benefit the deep learning of computer vision.

6.2.2.4 Improve the Feature Engineering of Speed

As discussed in Chapter 4.2.2, the difference between the Roadmap images at 00:03:00 and 00:08:00 are slight. Because the relationship of speed and value of the pixel is: Value of pixel = 255 - integer of speed. If we amplify the fluctuation of speed, like multiplying the speed to an index, we can enlarge the difference of Roadmap Images at different times.

6.2.2.5 Expectations of Researches in Future

In the technical aspect of this research, only one Geographical Image is generated because the research area is only in Chengdu City. In the future, more data from metropolises all over the world are expected to join the image fusion framework, which will not only enrich the transport dataset at a global scale but also improve the performance of the framework by keeping updating it.

In the practical aspect of this research, we figure out the failure of a single LSTM with no feature engineering and the success of Resnet with the image fusion framework in this task. The contrast testifies that in traffic forecast applications, the understanding and interpretation of features in datasets before building any machine learning algorithm is essential and vital for the complicated and diversiform traffic forecast tasks. To have a good understanding and interpretation of features in datasets, researchers should have a comprehensive understanding of both the dataset and the actual traffic scene.

6.2.2.6 Personal Comment

Although compared with the performance of historical average, the performance of the image fusion framework is proved to be a slightly successful innovation with a 9% advantage, but it took double or triple workload in the feature engineering part, making it not cost-effective in the application. Furthermore, can the image fusion framework be successful on other similar datasets? Because of the time limit of the master's thesis, this question is not studied. But as far as I know, among many traffic prediction researches that use deep learning, only a small part is applied. A very important reason is that the methods or networks used in some researches can only achieve a specific good result on the specific datasets. That is to say, even though the performance of the models in the researches is quite good, the models are not applicable when applied to other datasets, which means that the models do not have a generalized good performance in the application. The conclusion is that in the subsequent scientific research and innovation of researchers, doing a good job of risk control is also an important part of scientific research. During the communications between me and some Ph.D. students, I realize that an effective risk control action is: comprehensively searching the applicability of the aimed researches for several weeks or even months before the research. Emphasize the preparation and do not rush will build up my confidence against the fear of failure.

A. Appendix

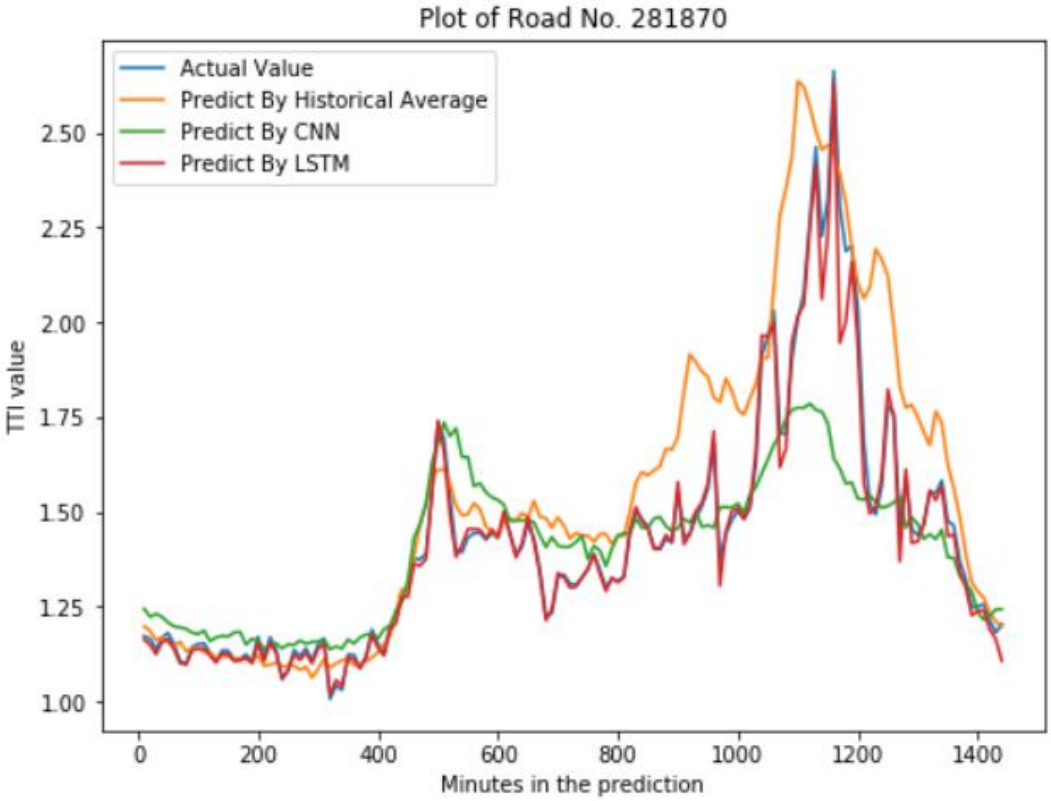


Figure A.1.: Performance of models in road No. 281870

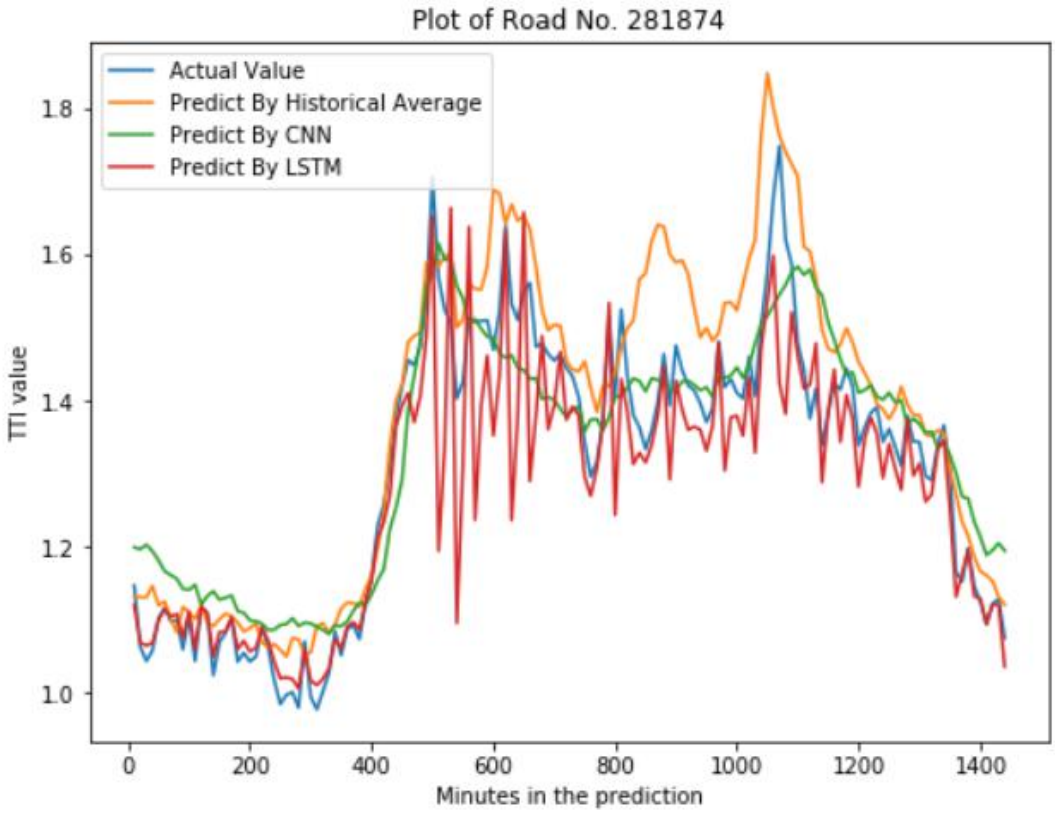


Figure A.2.: Performance of models in road No. 281874

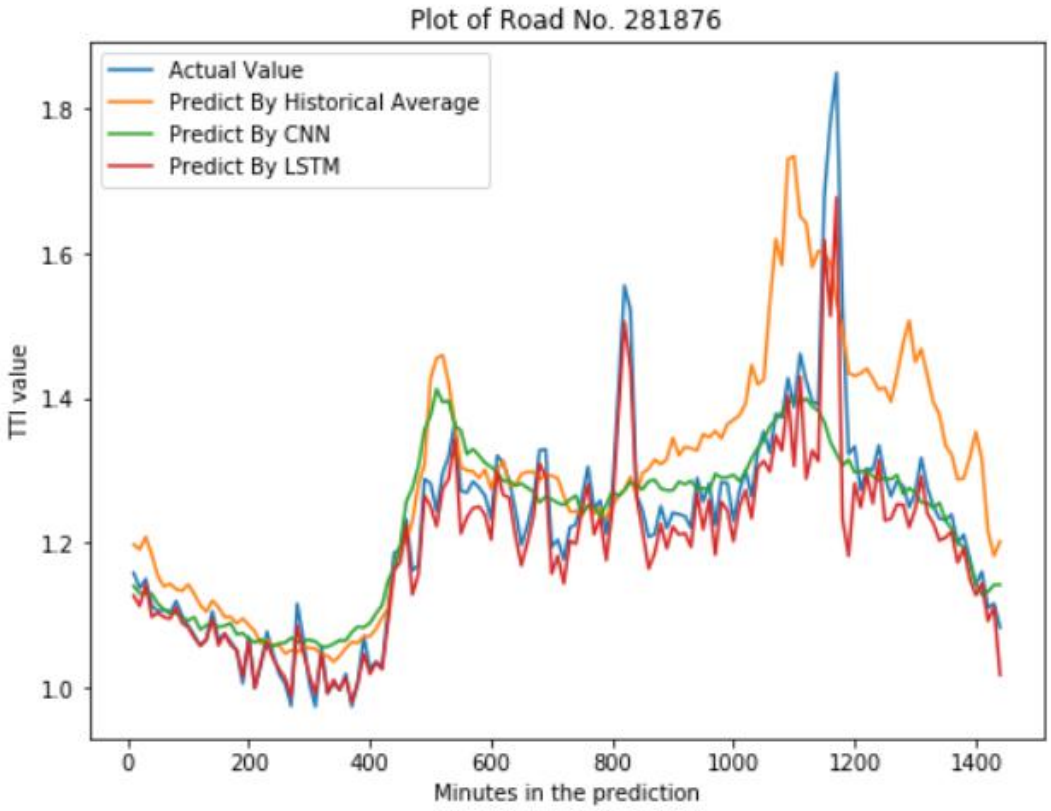


Figure A.3.: Performance of models in road No. 281876

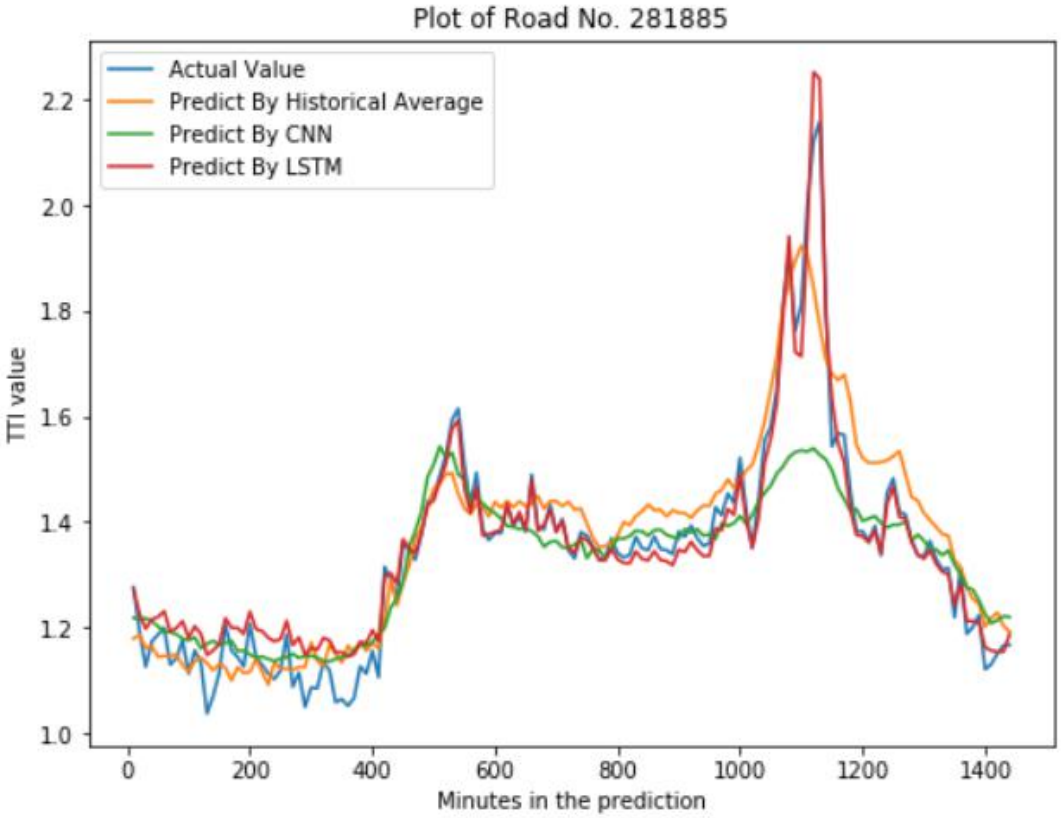


Figure A.4.: Performance of models in road No. 281885

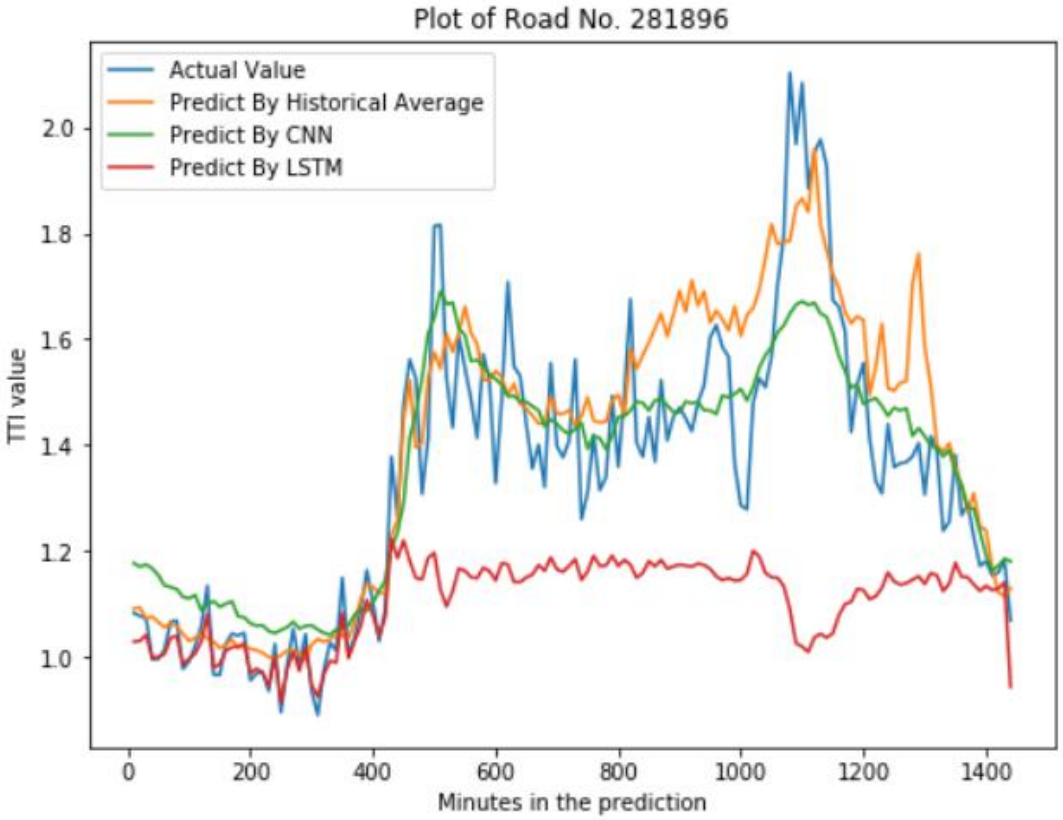


Figure A.5.: Performance of models in road No. 281896

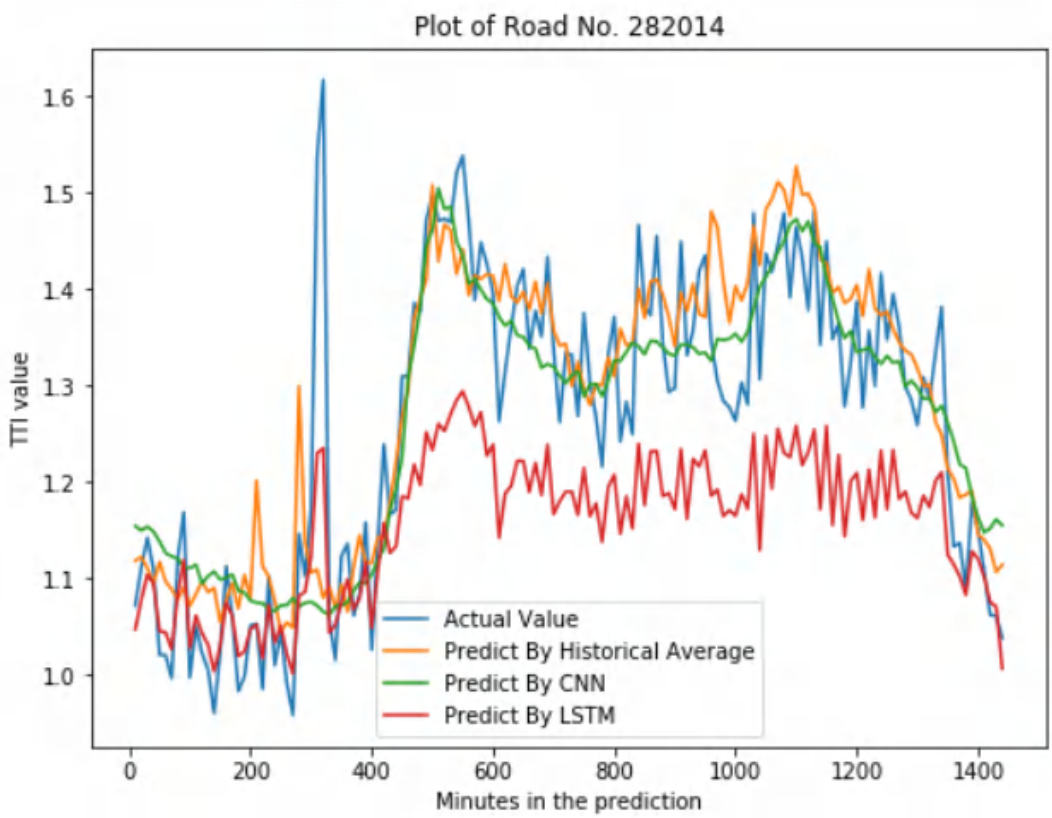


Figure A.6.: Performance of models in road No. 282014

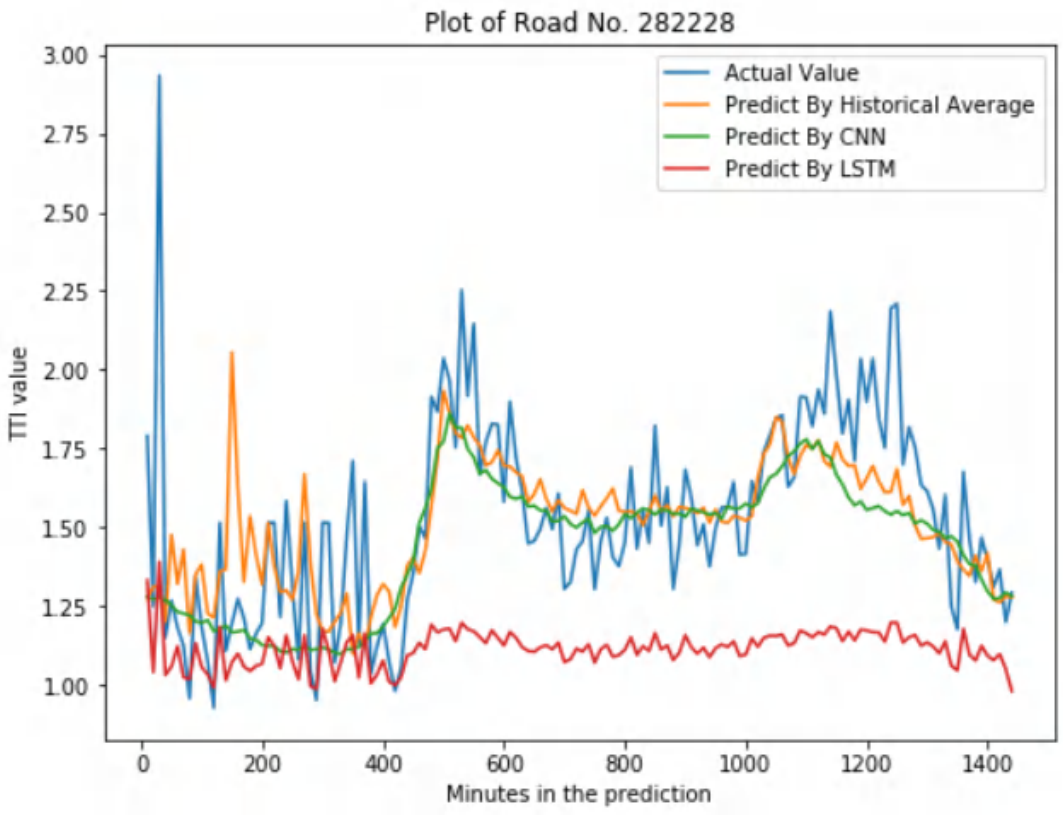


Figure A.7.: Performance of models in road No. 282228

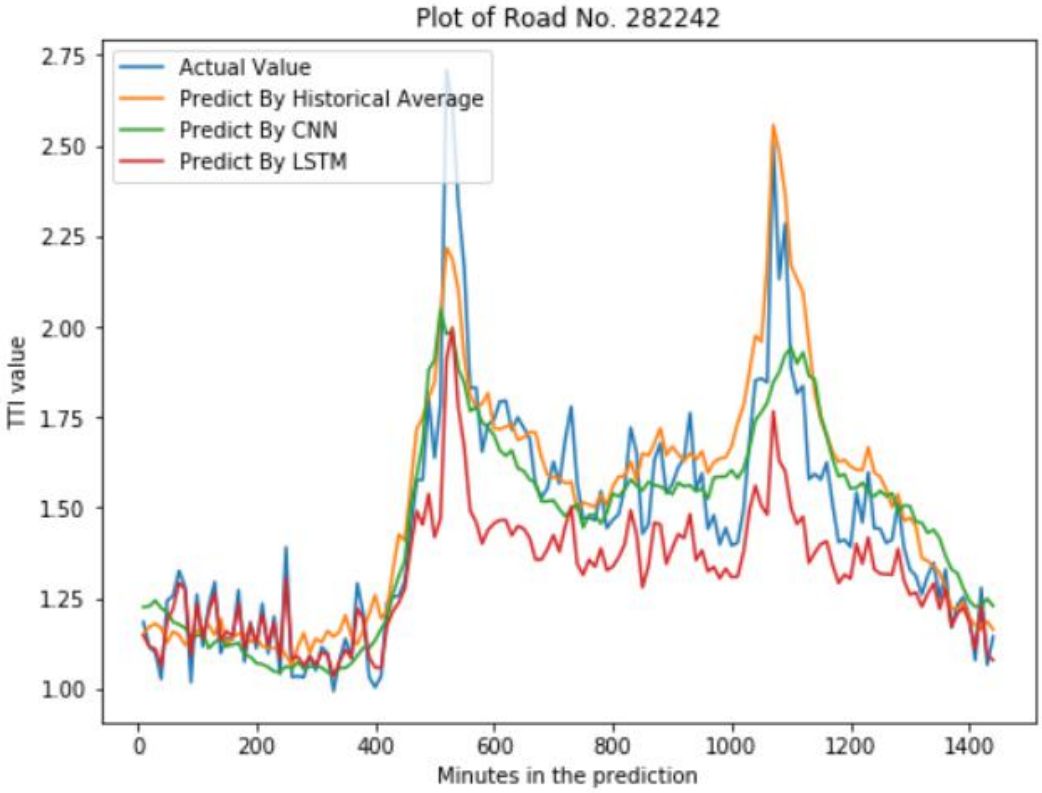


Figure A.8.: Performance of models in road No. 282242

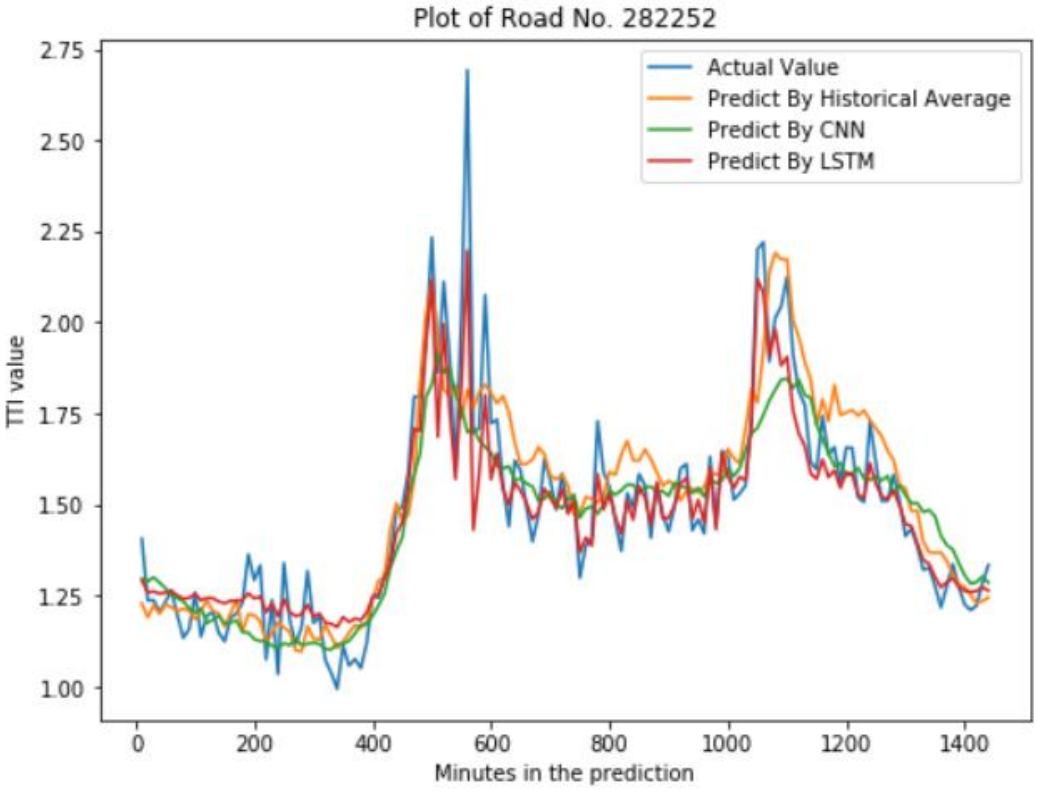


Figure A.9.: Performance of models in road No. 282252

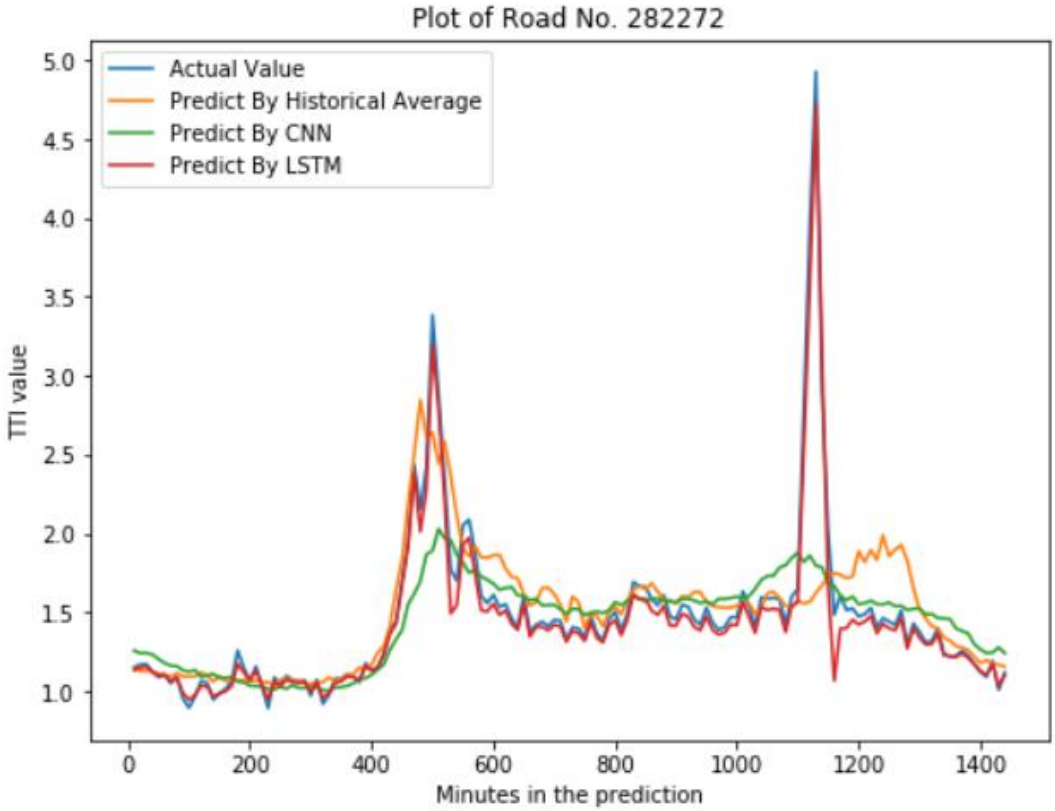


Figure A.10.: Performance of models in road No. 282272

```python
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
nowfile = filenames[64]
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

Nrow = len(pd.read_csv(os.path.join(dirname, nowfile),skiprows = 0,header = 0,usecols=[0,1,2,3],index_col = 0) )
raw_data=pd.read_csv(os.path.join(dirname, nowfile),skiprows = 0,header = 0,usecols=[0,1,2,3],index_col = 0,iterator = True)
long_lowerbound = 104.0402
lati_lowerbound = 30.6516
long_upperbound = 104.1298
lati_upperbound = 30.7284
x = [0,0,896,896,0]
y = [0,768,768,0,0]
plt.rcParams['figure.dpi'] = 128
plt.rcParams['savefig.dpi'] = 128
plt.rcParams['figure.figsize']=(7,6)
plt.rcParams['image.cmap'] = 'gray'
plt.figure()
plt.subplots_adjust(left = -0.05, bottom = -0.05, right = 1.05, top = 1.05, hspace = 0, wspace = 0)
plt.plot(x,y,color = '#000000', linewidth = 0.3)

for i in range(Nrow):
    print('Now process: ',i,' in /',Nrow)
    speedmatrix = []
    cd = raw_data.get_chunk(1)
    Splitdf = pd.DataFrame([],columns = ['OriginalID','Longitude','Latitude','Timestamp'])
    IDList = cd['Trajectory'].iloc[:].index.tolist()
    for n, ele in enumerate(cd['Trajectory']):
        element = ele.strip('[]').split(',')
        for p in range(len(element)):
            elemen = element[p].split(' ')
            elem = [element[p] for element[p] in elemen if element[p]]
            Longdata = elem[0]
            Latidata = elem[1]
            Timedata = elem[2]
            IDdata = IDList[n]
            Split_n = pd.DataFrame({'OriginalID':IDdata,'Longitude':Longdata,'Latitude':Latidata,'Timestamp':Timedata},index = [p])
            Splitdf = Splitdf.append(Split_n,ignore_index = False)
    for k in range(len(Splitdf)-1):
        long0 = float(Splitdf['Longitude'].iloc[k+0])
        lati0 = float(Splitdf['Latitude'].iloc[k+0])
        long1 = float(Splitdf['Longitude'].iloc[k+1])
        lati1 = float(Splitdf['Latitude'].iloc[k+1])
        time0 = float(Splitdf['Timestamp'].iloc[k+0])
```

Figure A.11.: Python Code of Plotting Trajectory Images(Part 1)

```python
        time1 = float(Splitdf['Timestamp'].iloc[k+1])
        if (long0 > long_lowerbound) and (long0 < long_upperbound) and (lati0 > lati_lowerbound) and (lati0 < lati_upperbound) and
        (time0 > int(nowfile[0:10])) and (time0 < int(nowfile[11:21])):
            Xcoord0 = round(10000*(long0-long_lowerbound))
            Xcoord1 = round(10000*(long1-long_lowerbound))
            Ycoord0 = round(10000*(lati0-lati_lowerbound))
            Ycoord1 = round(10000*(lati1-lati_lowerbound))
            Xpoint = [Xcoord0,Xcoord1]
            Ypoint = [Ycoord0,Ycoord1]
            if ((long1-long0)**2+(lati1-lati0)**2) == 0:
                speed1 = 0
            else:
                speed1 = ((long1-long0)**2+(lati1-lati0)**2)**0.5/(time1-time0)*111.19*3600
                speed1 = round(speed1)
                if (int(speed1) > 150):
                    speed1 = 150.0
                else:
                    pass
            hexn = hex(255-int(speed1))
            hexnum = (hexn[2:4] if len(hexn) == 4 else ('0'+hexn[2:3]))
            if hexnum == '0x':
                print('Problem: ', long1,long0,lati1,lati0,time1,time0,speed1,i,hexn)
                break
            Plotcolor = str('#'+hexnum+hexnum+hexnum)
            plt.plot(Xpoint,Ypoint,color = Plotcolor, linewidth = 0.03)
        else:
            pass
plt.axis('off')
savename = nowfile[0:-4]+'.png'
```

Figure A.12.: Python Code of Plotting Trajectory Images(Part 2)

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from shapely.geometry import MultiLineString
from shapely.wkt import dumps, loads
import time, datetime
import os
import gc
initial_timeStamp = 1539273600+600*140
long_lowerbound = 104.0402
lati_lowerbound = 30.6516
long_upperbound = 104.1298
lati_upperbound = 30.7284
Newdf = pd.DataFrame([],columns = ['ID','Longitude','Latitude','LineN'])
with open('../Data/Section_201810192021/road_boundary.txt','r',encoding = 'utf-8') as f:
    data = f.readlines()
    for i in range(len(data)-1):
        listdata = data[i+1].strip('\n').split('\t')
        MLS = loads(listdata[2])
        for n,g in enumerate(MLS.geoms):
            for c in g.coords:
                if (c[0]>long_lowerbound) and (c[0]<long_upperbound) and (c[1]<lati_upperbound) and(c[1]>lati_lowerbound):
                    new = pd.DataFrame({'ID':int(listdata[0]),'Longitude':c[0],'Latitude':c[1],'LineN':n},index = [i])
                    Newdf = Newdf.append(new,ignore_index = False)
                else:
                    pass
tticsv = pd.read_csv('../Data/2018-10-01-00_2018-11-30-00_tti.csv',header = 0,index_col = 'Unnamed: 0')
tticsv.columns = ['ID','datetime','TTI','AVGSpeed']

for s in range(4):
    print('Now process: ',s)
    if s == 20:
        gc.collect()
        print('gc.collect')
    now_timeStamp = initial_timeStamp + 600*s
    timeArray = time.localtime(now_timeStamp)
    otherStyleTime1 = time.strftime("%Y/%m/%d %H:%M:%S",timeArray)
    otherStyleTime2 = time.strftime("%Y-%m-%d %H:%M:%S",timeArray)
    otherStyleTime3 = time.strftime("%Y-%m-%d-%H-%M-%S",timeArray)
    mincsv = tticsv[tticsv['datetime'] == otherStyleTime2]
    TTIarray = np.array(mincsv['TTI'])
    AVGSpeedarray = np.array(mincsv['AVGSpeed'])
    MedianAvgSpeed = np.median(AVGSpeedarray)
    MedianTTIarray = np.median(TTIarray)
    Mergedf = Newdf.merge(mincsv,on = 'ID',how = 'left')
    Mergedf = Mergedf.fillna({'datetime':otherStyleTime1,'TTI':MedianTTIarray,'AVGSpeed':float(round(MedianAvgSpeed))})
    Drawdf = pd.DataFrame([],columns = ['Draworder','Xcoord','Ycoord','Intspd','TTI'])
    Order = 0
```

Figure A.13.: Python Code of Plotting Roadmap Images(Part 1)

X in CAD	Y in CAD	X in MAP	Y in MAP
48252.08	32971.39	104.06568	30.662403
48107.21	32682.49	104.064247	30.659953
48415.06	32684.95	104.067406	30.66
47757.81	33442.65	104.060467	30.666796
49224.95	29761.95	104.076163	30.633734
49882.53	30118.02	104.082955	30.636845
50435.6294	30437.1438	104.088727	30.639868
50508.6315	30475.6904	104.089446	30.640214
50888.12	31074.15	104.093437	30.645609
51145.67	31540.69	104.096039	30.649874
51387.03	34015.6	104.098351	30.67224
50596.41	33861.98	104.090216	30.670878
49951.11	35150.81	104.083162	30.682422
49526.52	35313.51	104.078849	30.683866
50118.22	35457.97	104.085055	30.685167
49009.1719	35526.8479	104.073237	30.685778
49338.4	36268.05	104.076764	30.692345
48277.96	35311.73	104.065767	30.683672
48139.16	35181.98	104.064308	30.682551
47862.74	35063.63	104.061411	30.681342
47640.8	35558.4	104.059051	30.685845
46984.05	35427.35	104.052334	30.684424
46336.02	35483.38	104.045414	30.684996
46270.08	35129.1	104.044754	30.681804
45927.16	34480.23	104.041182	30.675843
45862.476	33385.2593	104.040618	30.666117
51113.39	33916.57	104.095422	30.67141
51757.8093	32862.3970	104.102449	30.6619
52135.88	32845.06	104.106215	30.661734
51770.49	32861.37	104.102471	30.6619
52110.94	32377.89	104.106043	30.657525
51379.81	31975.42	104.098463	30.653833
48721.42	33459.92	104.070638	30.667073

Table A.1.: X and Y coordinates of corresponding landmarks in both CAD image and MAP image

```
49
50      for i in range(len(Mergedf)-1):
51          Draworder_item = Order
52          Xcoord = int(round(10000*(Mergedf['Longitude'][i]-long_lowerbound)))
53          Ycoord = int(round(10000*(Mergedf['Latitude'][i]-lati_lowerbound)))
54          Intspd = float(round(Mergedf['AVGSpeed'][i]))
55          newtti = Mergedf['TTI'][i]
56          draw_n = pd.DataFrame({'Draworder':Order,'Xcoord':Xcoord,'Ycoord':Ycoord,'Intspd':Intspd,'TTI':newtti},index = [i])
57          Drawdf = Drawdf.append(draw_n,ignore_index = False)
58          if (Mergedf['ID'][i] == Mergedf['ID'][i+1]) and (Mergedf['LineN'][i] == Mergedf['LineN'][i+1]):
59              pass
60
61          else:
62              Order += 1
63      x = [0,0,896,896,0]
64      y = [0,768,768,0,0]
65      plt.rcParams['figure.dpi'] = 128
66      plt.rcParams['savefig.dpi'] = 128
67      plt.rcParams['figure.figsize']=(7,6)
68      plt.rcParams['image.cmap'] = 'gray'
69      plt.figure()
70      plt.subplots_adjust(left = -0.05, bottom = -0.05, right = 1.05, top = 1.05, hspace = 0, wspace = 0)
71      plt.plot(x,y,color = '#000000', linewidth = 0.3)
72      EndNum = Drawdf['Draworder'].iloc[-1]
73      for j in range(EndNum+1):
74          Segment = Drawdf.loc[Drawdf['Draworder'] == j]
75          Xpoint = np.array(Segment['Xcoord'])
76          Ypoint = np.array(Segment['Ycoord'])
77          Spdpixel = Segment['Intspd'].iloc[0]
78          hexn = hex(255-int(Spdpixel))
79          hexnum = (hexn[2:4] if len(hexn) == 4 else ('0'+hexn[2:3]))
80          Plotcolor = str('#'+hexnum+hexnum+hexnum)
81          plt.plot(Xpoint,Ypoint,color = Plotcolor, linewidth = 0.3)
82      plt.axis('off')
83      savestr =otherStyleTime3 + '.png'
84      plt.savefig(os.path.join('../Data/Phase1_result/Roadmap_Pic',savestr))
```

Figure A.14.: Python Code of Plotting Roadmap Images(Part 2)

```python
#Part of codes are refer from https://www.kaggle.com/huanvo/lyft-complete-train-and-prediction-pipeline by the author Huan Vo in Kaggle
from typing import Dict
from sklearn.model_selection import KFold
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import torch
from torch import nn, optim
from torch.utils.data import DataLoader
from torchvision.models.resnet import resnet50, resnet18, resnet34, resnet101
from tqdm import tqdm
from torch.optim.lr_scheduler import ReduceLROnPlateau
from pathlib import Path
from PIL import Image
import matplotlib.pyplot as plt
from torch.utils.data import Dataset
from torchvision import transforms, utils
import os
import random
import time
import warnings
import math
warnings.filterwarnings("ignore")
os.environ['CUDA_VISIBLE_DEVICE'] = '1,4,5'

def set_seed(seed):
    random.seed(seed)
    np.random.seed(seed)
    os.environ["PYTHONHASHSEED"] = str(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
set_seed(42)

cfg = {

    'data_path': "./picture",
    'model_params': {
        'model_architecture': 'resnet34',
        'model_name': "model_resnet34_output",
        'lr': 1e-3,
    },
    'train_params': {
        'max_num_steps':160 ,
        'checkpoint_every_n_steps':40,
        'epoch':90
    }
}
```

Figure A.15.: Python Code of Resnet34 Training(Part 1)

```
data_transforms = transforms.Compose([
        transforms.Pad(padding = (0,64), fill=0, padding_mode='constant'),
    #   transforms.Resize((896,896)),
        transforms.ToTensor(),
    ])
total_df = pd.read_csv('./csvdata/Lasttotal_4th.csv')
train_valid_df = total_df.iloc[2:3458].copy()
train_valid_df['Time'] = list(total_df['Time'].iloc[1:3457])
test_df = total_df.iloc[3458:3602].copy()
test_df['Time'] = list(total_df['Time'].iloc[3457:3601])
for u in range(len(train_valid_df)):
    tssl = train_valid_df['Time'].iloc[u]+':00'
    timearray = time.strptime(tssl,"%Y/%m/%d %H:%M:%S")
    otherStyleTime = time.strftime("%Y_%m_%d-%H-%M-%S",timearray)
    train_valid_df['Time'].iloc[u] = otherStyleTime+'.png'

for k in range(len(test_df)):
    tssl = test_df['Time'].iloc[k]+':00'
    timearray = time.strptime(tssl,"%Y/%m/%d %H:%M:%S")
    otherStyleTime = time.strftime("%Y_%m_%d-%H-%M-%S",timearray)
    test_df['Time'].iloc[k] = otherStyleTime+'.png'

class Mydataset(Dataset):
    def __init__(self,df_data,data_dir='./picture',transform=data_transforms):
        super().__init__()
        self.df=df_data
        self.data_dir=data_dir
        self.transform=transform
    def __len__(self):
        return len(self.df)
    def __getitem__(self,idex):
        img_name=self.df['Time'].iloc[idex]
        label=self.df.copy().drop(['Time'],axis = 1).iloc[idex]
        img_path=os.path.join(self.data_dir,img_name)
        img = Image.open(img_path)
        img_tensor = self.transform(img)
        label = np.array(label)
        return img_tensor,label
train_valid_dataset = Mydataset(df_data = train_valid_df,data_dir='./picture/',transform = data_transforms)
test_dataset = Mydataset(df_data = test_df,data_dir='./testpic/',transform = data_transforms)
test_loader = DataLoader(dataset=test_dataset,batch_size=1,shuffle=False)
```

Figure A.16.: Python Code of Resnet34 Training(Part 2)

```python
class TransportModel(nn.Module):
    def __init__(self, cfg: Dict):
        super().__init__()
        architecture = cfg["model_params"]["model_architecture"]
        backbone = eval(architecture)(pretrained=False, progress=True)
        self.backbone = backbone
        if architecture == "resnet50":
            backbone_out_features = 2048
        else:
            backbone_out_features = 512
        self.head = nn.Sequential(
            nn.Linear(in_features=backbone_out_features, out_features=4096),
        )
        self.logit = nn.Linear(4096, out_features=11)
    def forward(self, x):
        x = self.backbone.conv1(x)
        x = self.backbone.bn1(x)
        x = self.backbone.relu(x)
        x = self.backbone.maxpool(x)
        x = self.backbone.layer1(x)
        x = self.backbone.layer2(x)
        x = self.backbone.layer3(x)
        x = self.backbone.layer4(x)
        x = self.backbone.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.head(x)
        x = self.logit(x)
        return x

def forward(inputs, targets,model, device, criterion = nn.L1Loss()):
    inputs = inputs.to(device)
    targets = targets.to(device)
    preds = model(inputs)
    loss = criterion(preds.float(),targets.float())
    return loss, preds

def _val(loader, model,device,criterion = nn.L1Loss()):
    model.eval()
    valid_preds = np.zeros((len(loader.dataset), 11))
    avg_val_loss = 0.0
    for i, (i_batch, y_batch) in enumerate(loader):
        i_batch = i_batch.to(device)
        y_batch = y_batch.to(device)
        with torch.no_grad():
            y_pred = model(i_batch).detach()
            avg_val_loss += criterion(y_pred.float(), y_batch.float()).item() / len(loader)
    return valid_preds, avg_val_loss
```

Figure A.17.: Python Code of Resnet34 Training(Part 3)

```python
def _test(loader, model,device,criterion = nn.L1Loss()):
    model.eval()
    valid_preds = np.zeros((len(loader.dataset), 11))
    avg_val_loss = 0.0
    for i, (i_batch, y_batch) in enumerate(loader):
        i_batch = i_batch.to(device)
        y_batch = y_batch.to(device)
        with torch.no_grad():
            y_pred = model(i_batch).detach()
            avg_val_loss += criterion(y_pred.float(), y_batch.float()).item() / len(loader)
            valid_preds[i] = y_pred.cpu().numpy()
    return valid_preds, avg_val_loss

kf = KFold(n_splits=6, random_state=43, shuffle=False)
n = 0
for train_index, valid_index in kf.split(train_valid_dataset):
    n += 1
    print("K-Fold cross-validation, now is:", n,'/6')
    X_train, X_valid = train_valid_df.iloc[train_index], train_valid_df.iloc[valid_index]
    train_dataset = Mydataset(df_data = X_train,data_dir='./picture/',transform = data_transforms)
    valid_dataset = Mydataset(df_data = X_valid,data_dir='./picture/',transform = data_transforms)
    train_loader = DataLoader(dataset=train_dataset,batch_size=18,shuffle=False)
    valid_loader = DataLoader(dataset=valid_dataset,batch_size=4,shuffle=False)
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = TransportModel(cfg)
    weight_path = './model_resnet34_output_'+str(n)+'_fold.pth'
    if weight_path:
        print('load weight path: ',weight_path)
        model.load_state_dict(torch.load(weight_path))
    model.to(device)
    optimizer = optim.Adam(model.parameters(), lr=cfg["model_params"]["lr"])
    scheduler = ReduceLROnPlateau(optimizer, 'min',factor=0.3,verbose=1,patience=8)
    print(f'device {device}')
    tr_it = iter(train_loader)
    epoch = cfg["train_params"]["epoch"]
    min_val_loss = 100.0
    for k in range(epoch):
        progress_bar = tqdm(range(cfg["train_params"]["max_num_steps"]))
        num_iter = cfg["train_params"]["max_num_steps"]
        losses_train = []
        iterations = []
        metrics = []
        times = []
        model_name = cfg["model_params"]["model_name"]
        start = time.time()
        print('Begin Epoch: ',k)
        for i in progress_bar:
            try:
```

Figure A.18.: Python Code of Resnet34 Training(Part 4)

```
                images,labels = tr_it.next()
            except StopIteration:
                tr_it = iter(train_loader)
                images,labels = tr_it.next()
            model.train()
            torch.set_grad_enabled(True)
            loss, _ = forward(images,labels, model, device)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            losses_train.append(loss.item())
            progress_bar.set_description(f"train_loss: {loss.item()} train_loss(avg): {np.mean(losses_train)}")
            if i % cfg['train_params']['checkpoint_every_n_steps'] == 0:
                iterations.append(i)
                metrics.append(np.mean(losses_train))
                times.append((time.time()-start)/60)
        valid_preds, avg_val_loss = _val(valid_loader, model,device)
        scheduler.step(avg_val_loss)
        if avg_val_loss < min_val_loss:
            min_val_loss = avg_val_loss
            print('Renew Validation Loss')
            torch.save(model.state_dict(), f'{model_name}_{n}_fold.pth')
        else:
            pass
        print('val_loss: ',avg_val_loss)
    results = pd.DataFrame({'KFold': n,'Epoch': k,'iterations': iterations,
    'metrics (avg)': metrics, 'elapsed_time (mins)': times})
    print(f"Total training time is {(time.time()-start)/60} mins")
    print('Minimum Validation Loss is: ',min_val_loss)
    print(results.head())

Lastresult = np.zeros((test_dataset.__len__(), 11))
for o in range(6):
    print('Predict test fold: ',o+1)
    modelx = TransportModel(cfg)
    modelx.to(device)
    path = 'model_resnet34_output_'+str(o+1)+'_fold.pth'
    modelx.load_state_dict(torch.load(path))
    test_preds_foldx, avg_test_loss_foldx = _test(test_loader, modelx,device)
    Lastresult += (test_preds_foldx)/6
result_df = test_df.copy()
for b,w in enumerate(test_df.columns.tolist()[1:]):
    result_df[w] = Lastresult[:,b]
result_df.to_csv('CNN_resnet34_6fold.csv')
target = np.array(test_df.copy().drop(['Time'],axis = 1))
print('Total Loss of test file: ',np.sum(abs(target - Lastresult)))
```

Figure A.19.: Python Code of Resnet34 Training(Part 5)

```python
import torch
import torch.nn as nn
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import KFold
from torch.optim.lr_scheduler import ReduceLROnPlateau
from sklearn.preprocessing import MinMaxScaler

raw_total_df = pd.read_csv('/kaggle/input/csvdata2/Lasttotal_4th.csv')
total_df = raw_total_df.iloc[2:3602].copy()
total_df['Time'] = list(raw_total_df['Time'].iloc[1:3601])
road_id = total_df.columns[11]
fig_size[0] = 15
fig_size[1] = 5
plt.rcParams["figure.figsize"] = fig_size
plt.title('Time vs TTI in road No.'+road_id)
plt.ylabel('TTI')
plt.xlabel('Time per 10 min')
plt.grid(True)
plt.autoscale(axis='x',tight=True)
plt.plot(total_df[road_id])
all_data = total_df[road_id].values.astype(float)
test_data_size = 144
train_data = all_data[:-test_data_size]
test_data = all_data[-test_data_size:]
scaler = MinMaxScaler(feature_range=(-1, 1))
train_data_normalized = scaler.fit_transform(train_data .reshape(-1, 1))
train_data_normalized = torch.FloatTensor(train_data_normalized).view(-1)
train_window = 144
def create_inout_sequences(input_data, tw):
    inout_seq = []
    L = len(input_data)
    for i in range(L-tw):
        train_seq = input_data[i:i+tw]
        train_label = input_data[i+tw:i+tw+1]
        inout_seq.append((train_seq ,train_label))
    return inout_seq

class LSTM(nn.Module):
    def __init__(self, input_size=1, hidden_layer_size=100, output_size=1):
        super().__init__()
        self.hidden_layer_size = hidden_layer_size
        self.lstm = nn.LSTM(input_size, hidden_layer_size)
        self.linear = nn.Linear(hidden_layer_size, output_size)
        self.hidden_cell = (torch.zeros(1,1,self.hidden_layer_size),
                            torch.zeros(1,1,self.hidden_layer_size))
```

Figure A.20.: Python Code of LSTM with Single Input Training(Part 1)

```
50
51      def forward(self, input_seq):
52          lstm_out, self.hidden_cell = self.lstm(input_seq.view(len(input_seq) ,1, -1), self.hidden_cell)
53          predictions = self.linear(lstm_out.view(len(input_seq), -1))
54          return predictions[-1]
55  device = torch.device("cpu")
56  model = LSTM()
57  epochs = 30
58  criterion = nn.L1Loss()
59
60  valid_preds = np.zeros((576, 1))
61  avg_val_loss = 0.0
62  kf = KFold(n_splits=6, shuffle=False)
63  n = 0
64  for train_index, valid_index in kf.split(train_data):
65      model = LSTM()
66      loss_function = nn.L1Loss()
67      optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
68      scheduler = ReduceLROnPlateau(optimizer, 'min',factor = 0.3,verbose=1,patience=3)
69      n += 1
70      print("K-Fold cross-validation, now is:", n,'/6')
71      X_train, X_valid = train_data[train_index], train_data[valid_index]
72      scaler = MinMaxScaler(feature_range=(-1, 1))
73      X_train_data_normalized = scaler.fit_transform(X_train .reshape(-1, 1))
74      X_valid_data_normalized = scaler.fit_transform(X_valid .reshape(-1, 1))
75      X_train_data_normalized = torch.FloatTensor(X_train_data_normalized).view(-1)
76      X_valid_data_normalized = torch.FloatTensor(X_valid_data_normalized).view(-1)
77      X_train_inout_seq = create_inout_sequences(X_train_data_normalized, train_window)
78      X_valid_inout_seq = create_inout_sequences(X_valid_data_normalized, train_window)
79      minloss = 100.0
80      for i in range(epochs):
81
82          valid_preds = np.zeros((576, 1))
83          avg_val_loss = 0.0
84          for seq, labels in X_train_inout_seq:
85              optimizer.zero_grad()
86              model.hidden_cell = (torch.zeros(1, 1, model.hidden_layer_size),
87                                   torch.zeros(1, 1, model.hidden_layer_size))
88              model.to(device)
89              seq = seq.to(device)
90              labels = labels.to(device)
91              y_pred = model(seq)
92              single_loss = loss_function(y_pred, labels)
93              single_loss.backward()
94              optimizer.step()
95          for seq, labels in X_valid_inout_seq:
96              optimizer.zero_grad()
97              with torch.no_grad():
```

Figure A.21.: Python Code of LSTM with Single Input Training(Part 2)

```
                y_pred = model(seq).detach()
                avg_val_loss += criterion(y_pred.float(), labels.float()).item() / (len(X_valid_inout_seq)-test_data_size)
        if avg_val_loss < minloss:
            minloss = avg_val_loss
            print('Renew Validation Loss: ',avg_val_loss)
            torch.save(model.state_dict(), f'lstm_{n}_fold.pth')
        else:
            pass
        scheduler.step(avg_val_loss)

model.eval()
Lastresult = np.zeros((144, 1))
#------If you use predicted data to do prediction, use the following part
for o in range(6):
    print('Predict test fold: ',o+1)
    modelx = LSTM()
    modelx.to(device)

    path = 'lstm_'+str(o+1)+'_fold.pth'
    modelx.load_state_dict(torch.load(path))
    step_tensor = train_data_normalized[-train_window:]
    midresult = np.zeros((144, 1))
    if True:
        for s in range(144):
            optimizer.zero_grad()
            with torch.no_grad():
                y_pred = model(step_tensor).detach()
                midresult[s] = y_pred
                avg_val_loss += criterion(y_pred.float(), labels.float()).item() / 144
                step_tensor = torch.hstack((step_tensor,y_pred))[1:]
    actual_predictions = scaler.inverse_transform(midresult ).reshape(-1, 1)
    Lastresult += (actual_predictions)/6
print(Lastresult)
#------End of this part

#------If you use test data to do prediction, use the following part
model.eval()
Lastresult = np.zeros((144, 1))
for o in range(6):
    print('Predict test fold: ',o+1)
    modelx = LSTM()
    modelx.to(device)
    path = 'lstm_'+str(o+1)+'_fold.pth'
    modelx.load_state_dict(torch.load(path))
    test_inputs = train_data_normalized[-train_window:]
    test_data_normalized = scaler.fit_transform(test_data .reshape(-1, 1))
    test_data_normalized = torch.FloatTensor(test_data_normalized).view(-1)
    lasttensor = torch.hstack((train_data_normalized[-train_window:],test_data_normalized))
```

Figure A.22.: Python Code of LSTM with Single Input Training(Part 3)

```
146         test_inout_seq = create_inout_sequences(lasttensor, train_window)
147         midresult = np.zeros((144, 1))
148         for i in range(144):
149             for s,(seq, labels) in enumerate(test_inout_seq):
150                 optimizer.zero_grad()
151                 with torch.no_grad():
152                     y_pred = model(seq).detach()
153                     midresult[s] = y_pred
154                     avg_val_loss += criterion(y_pred.float(), labels.float()).item() / 144
155         actual_predictions = scaler.inverse_transform(midresult ).reshape(-1, 1)
156         Lastresult += (actual_predictions)/6
157     print(Lastresult)
158     #-------End of this part
159
160     print('Total Loss of test file: ',np.sum(abs(test_data.reshape((144,1)) - Lastresult)))
161     x = np.arange(3457, 3601, 1)
162     plt.title('Time vs TTI')
163     plt.ylabel('TTI')
164     plt.grid(True)
165     plt.autoscale(axis='x', tight=True)
166     plt.plot(total_df[road_id])
167     plt.plot(x,Lastresult)
168     plt.show()
169
170     plt.title('Time vs TTI')
171     plt.ylabel('TTI')
172     plt.grid(True)
173     plt.autoscale(axis='x', tight=True)
174
175     plt.plot(total_df[road_id][-train_window:])
176     plt.plot(x,Lastresult)
177     plt.show()
178     np.savetxt('result'+road_id+'.txt',Lastresult)
```

Figure A.23.: Python Code of LSTM with Single Input Training(Part 4)

```python
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
from typing import Dict
from sklearn.model_selection import KFold
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import torch
from torch import nn, optim
from torch.utils.data import DataLoader
from torchvision.models.resnet import resnet50, resnet18, resnet34, resnet101
from tqdm import tqdm
import torch.nn.functional as F
from torch.optim.lr_scheduler import ReduceLROnPlateau
from pathlib import Path
from PIL import Image
import matplotlib.pyplot as plt
from torch.utils.data import Dataset
from torchvision import transforms, utils
import os
import random
import time
import warnings
from collections import OrderedDict
import math
warnings.filterwarnings("ignore")
import torch
import torch.nn as nn
%matplotlib inline

def set_seed(seed):
    random.seed(seed)
    np.random.seed(seed)
    os.environ["PYTHONHASHSEED"] = str(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
set_seed(42)
cfg = {
    'format_version': 4,
    'data_path': "/kaggle/input/csvdata2",
    'model_params': {
        'model_architecture': 'LSTM',
        'model_name': "LSTM",
        'lr': 1e-3,
        'weight_path': "/kaggle/input/resnet34/resnet34.pth",

    },
```

Figure A.24.: Python Code of LSTM with Multiple Input Training(Part 1)

```
50
51      'train_params': {
52          'max_num_steps':110 ,
53          'checkpoint_every_n_steps': 25,
54          'epoch':90
55      }
56  }
57
58  total_df = pd.read_csv('/kaggle/input/csvdata2/LSTM_try.csv')
59  total_df.iloc[36000:]['Name'] = '000000'
60  test_df = total_df[total_df['Month']==11].copy()
61  for h in range(11):
62      test_df = test_df.drop(index = 3455+h*3600)
63  set_diff_df = total_df.append(test_df)
64  set_diff_df = set_diff_df.drop_duplicates(keep=False)
65  class Mydataset(Dataset):
66      def __init__(self,df_data):
67          super().__init__()
68          self.df=df_data
69      def __len__(self):
70          return len(self.df)
71      def __getitem__(self,idex):
72          name=float(self.df['Name'].iloc[idex])
73          month=float(self.df['Month'].iloc[idex])
74          day=float(self.df['Day'].iloc[idex])
75          hour=float(self.df['Hour'].iloc[idex])
76          minute=float(self.df['Minute'].iloc[idex])
77          speed=self.df['Speed'].iloc[idex]
78          length=self.df['Length'].iloc[idex]
79          tti=self.df['TTI'].iloc[idex]
80          inputarrays = np.array([name,month,day,hour,minute,speed,length])
81          label = np.array([tti])
82          return inputarrays,label
83
84  train_valid_dataset = Mydataset(df_data =set_diff_df)
85  test_dataset = Mydataset(df_data = test_df)
86  train_valid_loader = DataLoader(dataset=train_valid_dataset,batch_size=1,shuffle=False)
87  test_loader = DataLoader(dataset=test_dataset,batch_size=1,shuffle=False)
```

Figure A.25.: Python Code of LSTM with Multiple Input Training(Part 2)

```
class LSTM(nn.Module):
    def __init__(self, input_size=7, hidden_layer_size=100, output_size=1):
        super().__init__()
        self.hidden_layer_size = hidden_layer_size
        self.lstm = nn.LSTM(input_size, hidden_layer_size)
        self.linear = nn.Linear(hidden_layer_size, output_size)
        self.hidden_cell = (torch.zeros(1,1,self.hidden_layer_size),
                            torch.zeros(1,1,self.hidden_layer_size))
    def forward(self, input_seq):
        lstm_out, self.hidden_cell = self.lstm(input_seq.view(len(input_seq) ,1, -1), self.hidden_cell)
        predictions = self.linear(lstm_out.view(len(input_seq), -1))
        return predictions[-1]
model = LSTM()
print(model)
def forward(inputs, targets,model, criterion = nn.L1Loss()):
    preds = model(inputs.float())
    loss = criterion(preds.float(),targets.float())
    return loss, preds

def _val(loader, model,criterion = nn.L1Loss()):

    model.eval()
    valid_preds = np.zeros((len(loader.dataset), 1))
    avg_val_loss = 0.0
    for i, (i_batch, y_batch) in enumerate(loader):
        i_batch = i_batch
        y_batch = y_batch
        with torch.no_grad():
            y_pred = model(i_batch.float()).detach()
            avg_val_loss += criterion(y_pred.float(), y_batch.float()).item() / len(loader)
    return valid_preds, avg_val_loss

def _test(loader, model,criterion = nn.L1Loss()):
    model.eval()
    valid_preds = np.zeros((len(loader.dataset), 1))
    avg_val_loss = 0.0
    for i, (i_batch, y_batch) in enumerate(loader):

        with torch.no_grad():
            y_pred = model(i_batch.float()).detach()
            y_pred = y_pred.resize(1,1)

            avg_val_loss += criterion(y_pred.float(), y_batch.float()).item() / len(loader)
            valid_preds[i] = y_pred.cpu().numpy()
    return valid_preds, avg_val_loss
criterion = nn.L1Loss()
kf = KFold(n_splits=6, random_state=42, shuffle=True)
n = 0
```

Figure A.26.: Python Code of LSTM with Multiple Input Training(Part 3)

```
for train_index, valid_index in kf.split(train_valid_dataset):
    n += 1
    print("K-Fold cross-validation, now is:", n,'/6')
    print("TRAIN:", train_index, "VALID:", valid_index)
    X_train, X_valid = total_df.iloc[train_index], total_df.iloc[valid_index]
    train_dataset = Mydataset(df_data = X_train)
    valid_dataset = Mydataset(df_data = X_valid)
    train_loader = DataLoader(dataset=train_dataset,batch_size=288,shuffle=True)
    valid_loader = DataLoader(dataset=valid_dataset,batch_size=60,shuffle=True)
    model = LSTM()
    optimizer = optim.Adam(model.parameters(), lr=cfg["model_params"]["lr"])
    scheduler = ReduceLROnPlateau(optimizer, 'min',factor = 0.3,verbose=1,patience=8)
    optimizer.zero_grad()
    model.hidden_cell = (torch.zeros(1, 1, model.hidden_layer_size),
                         torch.zeros(1, 1, model.hidden_layer_size))
    tr_it = iter(train_loader)
    epoch = cfg["train_params"]["epoch"]
    min_val_loss = 100.0
    for k in range(epoch):
        progress_bar = tqdm(range(cfg["train_params"]["max_num_steps"]))
        num_iter = cfg["train_params"]["max_num_steps"]
        losses_train = []
        iterations = []
        metrics = []
        times = []
        model_name = cfg["model_params"]["model_name"]
        start = time.time()
        print('Begin Epoch: ',k)
        for i in progress_bar:
            try:
                inputs,labels = tr_it.next()
            except StopIteration:
                tr_it = iter(train_loader)
                inputs,labels = tr_it.next()
            optimizer.zero_grad()
            model.hidden_cell = (torch.zeros(1, 1, model.hidden_layer_size),
                                 torch.zeros(1, 1, model.hidden_layer_size))
            y_pred = model(inputs.float())
            loss = criterion(y_pred, labels)
            loss.backward()
            optimizer.step()
            losses_train.append(loss.item())
            progress_bar.set_description(f"train_loss: {loss.item()} train_loss(avg): {np.mean(losses_train)}")
            if i % cfg['train_params']['checkpoint_every_n_steps'] == 0:
                iterations.append(i)
                metrics.append(np.mean(losses_train))
                times.append((time.time()-start)/60)

```

Figure A.27.: Python Code of LSTM with Multiple Input Training(Part 4)

```
183
184         valid_preds, avg_val_loss = _val(valid_loader, model)
185         scheduler.step(avg_val_loss)
186         if avg_val_loss < min_val_loss:
187             min_val_loss = avg_val_loss
188             print('Renew Validation Loss')
189             torch.save(model.state_dict(), f'{model_name}_{n}_fold.pth')
190         else:
191             pass
192         print('val_loss: ',avg_val_loss)
193         print(f"Total training time is {(time.time()-start)/60} mins")
194         print('Minimum Validation Loss is: ',min_val_loss)
195
196 Lastresult = np.zeros((test_dataset.__len__(),1))
197 for o in range(6):
198     print('Predict test fold: ',o+1)
199     modelx = LSTM()
200
201     path = 'LSTM_'+str(o+1)+'_fold.pth'
202     modelx.load_state_dict(torch.load(path))
203     test_preds_foldx, avg_test_loss_foldx = _test(test_loader, modelx)
204     Lastresult += (test_preds_foldx)/6
205
206 raw_df = pd.read_csv('/kaggle/input/csvdata2/Lasttotal_4th.csv')
207 previous_test_df = raw_df.iloc[3458:3602].copy()
208 previous_test_df['Time'] = list(raw_df['Time'].iloc[3457:3601])
209 result_df = previous_test_df.copy()
210 for b,w in enumerate(previous_test_df.columns.tolist()[1:]):
211     result_df[w] = Lastresult[0+144*b:144*(b+1)]
212 result_df.to_csv('LSTM_result.csv',index = False)
213 target = np.array(test_df['TTI'])
214 target = np.reshape(target,(1584,1))
215 print('Average Loss of test file: ',np.sum(abs(target - Lastresult))/1584)
216
217
```

Figure A.28.: Python Code of LSTM with Multiple Input Training(Part 5)

List of Figures

List of Tables

Bibliography

[1] *DIRECTIVE 2010/40/EU OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 7 July 2010 on the framework for the deployment of Intelligent Transport Systems in the field of road transport and for interfaces with other modes of transport*. Last accessed 16 May 2021. URL: https://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2010:207:0001:0013:EN:PDF.

[2] Z. Karami and R. Kashef. "Smart Transportation Planning: Data, Models, and Algorithms". In: *Transportation Engineering* 2 (July 2020), p. 100013. DOI: 10.1016/j.treng.2020.100013.

[3] C. Hinsbergen, J. Lint, and F. Sanders. "Short Term Traffic Prediction Models". In: *14th World Congress on Intelligent Transport Systems, ITS 2007* 7 (Nov. 2007).

[4] J. Walsh, N. O' Mahony, S. Campbell, A. Carvalho, L. Krpalkova, G. Velasco-Hernandez, S. Harapanahalli, and D. Riordan. "Deep Learning vs. Traditional Computer Vision". In: Apr. 2019. ISBN: 978-981-13-6209-5. DOI: 10.1007/978-3-030-17795-9_10.

[5] P. Klauke. *Importance of Feature Engineering methods*. Last accessed 16 May 2021. 2019. URL: https://towardsdatascience.com/importance-of-feature-engineering-methods-73e4c41ae5a3.

[6] A. Ng. *Machine learning and AI via Brain simulations*. Last accessed 16 May 2021. URL: https://ai.stanford.edu/~ang/slides/DeepLearning-Mar2013.pptx.

[7] *AutoDesk*. Last accessed 16 May 2021. URL: https://www.autodesk.com/solutions/cad-software.

[8] *cadmapper*. Last accessed 16 May 2021. URL: https://cadmapper.com/terms/.

[9] *A short guide to chinese coordinate system*. Last accessed 16 May 2021. URL: https://abstractkitchen.com/blog/a-short-guide-to-chinese-coordinate-system/.

[10] I. Open GIS Consortium. *OpenGIS Simple Features Specification For SQL*. Last accessed 16 May 2021. URL: http://portal.opengeospatial.org/files/?artifact_id=829.

[11] R. Brundritt. *Two New Data Modules for Bing Maps V7*. Last accessed 16 May 2021. URL: https://mapsys.info/tag/reader/.

[12] *Epoch and Unix Timestamp Conversion Tools*. Last accessed 16 May 2021. URL: https://www.unixtimestamp.com/.

[13] C. Bradshaw. *Converting UNIX Timestamps to Excel Dates*. Last accessed 16 May 2021. URL: https://www.isjw.uk/post/software/converting-unix-timestamps-to-excel-dates/.

[14] Y. Quan, Y. Tong, W. Feng, G. Dauphin, W. Huang, and M. Xing. "A Novel Image Fusion Method of Multi-Spectral and SAR Images for Land Cover Classification". In: *Remote Sensing* 12.22 (2020). ISSN: 2072-4292. DOI: 10.3390/rs12223801. URL: https://www.mdpi.com/2072-4292/12/22/3801.

[15] T. Wang. *TTI ‚Smart Transportation Industry Standard!* Last accessed 16 May 2021. URL: https://github.com/didi/TrafficIndex.

[16] Y. Lecun and Y. Bengio. "Convolutional Networks for Images, Speech, and Time-Series". In: Jan. 1995.

[17] M. Valueva, N. Nagornov, P. Lyakhov, G. Valuev, and N. Chervyakov. "Application of the residue number system to reduce hardware costs of the convolutional neural network implementation". In: *Mathematics and Computers in Simulation* 177 (2020), pp. 232–243. ISSN: 0378-4754. DOI: https://doi.org/10.1016/j.matcom.2020.04.031. URL: https://www.sciencedirect.com/science/article/pii/S0378475420301580.

[18] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition". In: June 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[19] R. Staudemeyer and E. Morris. *Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks*. Sept. 2019.

[20] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[21] I. Islek and S. G. Ögüdücü. "Use of LSTM for Short-Term and Long-Term Travel Time Prediction". In: *CIKM Workshops*. 2018.

[22] D. Berrar. "Cross-Validation". In: Jan. 2018. ISBN: 9780128096338. DOI: 10.1016/B978-0-12-809633-8.20349-X.

[23] D. Wilson and T. Martinez. "The need for small learning rates on large problems". In: vol. 1. Feb. 2001, 115–119 vol.1. ISBN: 0-7803-7044-9. DOI: 10.1109/IJCNN.2001.939002.

[24] Y. Bengio. "Practical recommendations for gradient-based training of deep architectures". In: *Arxiv* (June 2012).

[25] *DiDi Research Outreach Initiative*. Last accessed 16 May 2021. URL: https://outreach.didichuxing.com/app-outreach/about.

[26] *Travel Time Index*. Last accessed 16 May 2021. URL: https://outreach.didichuxing.com/appEn-vue/TTI.

[27] *Travel Time Index and Trajectory*. Last accessed 16 May 2021. URL: https://outreach.didichuxing.com/app-vue/TTItrajectory.

[28] OpenStreetMaps. *Chengdu City, China*. Last accessed 16 May 2021 [Online]. URL: https://www.openstreetmap.org/relation/2110264#map=13/30.6393/104.0761.

[29] *How much distance does a degree, minute, and second cover on your maps?* Last accessed 16 May 2021. URL: https://www.usgs.gov/faqs/how-much-distance-does-a-degree-minute-and-second-cover-your-maps?qt-news_science_products=0#qt-news_science_products.

[30] *Urban Street Design Guide*. Last accessed 16 May 2021. URL: https://nacto.org/publication/urban-street-design-guide/street-design-elements/lane-width/.

[31] *Urban speed limit range*. Last accessed 16 May 2021. URL: https://www.who.int/gho/road_safety/legislation/situation_trends_urban_speed_limit/en/.

[32] GoogleMaps. *Chongqing City, China*. Last accessed 16 May 2021 [Online]. URL: https://www.google.com.my/maps/@29.5562811,106.5640351,12z.

[33] J. Zhang, Y. Zheng, and D. Qi. "Deep Spatio-Temporal Residual Networks for Citywide Crowd Flows Prediction". In: (Sept. 2016).

[34] *Gaode AI Platform*. Last accessed 16 May 2021. URL: https://lbs.amap.com/console/show/picker.

[35] Strava. *Strava Global Heatmap*. Last accessed 16 May 2021. URL: https://www.strava.com/heatmap#12.00/104.01781/30.66105/hot/all.

[36] *Dilapksy Lee's Kaggle Profile*. Last accessed 16 May 2021. URL: https://www.kaggle.com/dilapsky.

[37] *Torch, A scientific computing framework for LuaJIT*. Last accessed 16 May 2021. URL: http://torch.ch/.

[38] *Pytorch*. Last accessed 16 May 2021. URL: https://github.com/pytorch/pytorch.

[39] T. Shah. *About Train, Validation and Test Sets in Machine Learning*. Last accessed 16 May 2021. URL: https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7.

[40] A. Shekhar. *What Are L1 and L2 Loss Functions?* Last accessed 16 May 2021. URL: https://afteracademy.com/blog/what-are-l1-and-l2-loss-functions.

[41] *TORCH.OPTIM*. Last accessed 16 May 2021. URL: https://pytorch.org/docs/stable/optim.html#torch.optim.lr_scheduler.ReduceLROnPlateau.

[42] U. Malik. *Time Series Prediction using LSTM with PyTorch in Python*. Last accessed 16 May 2021. URL: https://stackabuse.com/time-series-prediction-using-lstm-with-pytorch-in-python.

[43] A. O. Fatola. *On-demand Modeling and Forecasting Individual Upcoming Trips*. 2020.