Abstract

In urban planning and management, traffic prediction is crucial for reducing congestion, optimizing urban layout, and estimating travel time throughout a city. The use of big data and AI (Artificial Intelligence) has revolutionized this field in recent years, transforming how traffic data is collected, analyzed, and utilized. It allowed the collection of a significant volume of data with great diversity, which is essential for AI models, known for being data-hungry. This optimization of data consumption enables computationally viable models.

Although state-of-the-art deep learning models can be highly accurate, they require large amounts of data to function correctly. This drawback, also known as the “cold-start” problem, can hinder cities from building their intelligent network, since inputting a certain amount of data is required to make the model functional. Recent studies propose using transfer learning mechanisms to handle this situation. These models can learn complex ST (Spatio-Temporal) patterns from data-rich cities and use this knowledge to make predictions in data-scarce counterparts. This approach can also reduce the computational burden of re-training models, as the model developed to extract complex patterns is reused even for other data-rich cities.

In this work, we explore the study and adaption of state-of-the-art convolutional graph models in a transfer learning framework to leverage multiple cities as sources of knowledge on traffic patterns. To achieve this, we utilized a substantially big dataset composed of eight cities. Specifically, we examined how the diversity of source data, through the use of multiple source cities, affected the model’s performance. The study compared the performance of several models to a baseline consisting of two statistical, and one deep learning models. To verify the accuracy of both the models and baselines, we employed several metrics, such as MAE (Mean Absolute Error), RMSE (Root Mean Squared Error), and MSE (Mean Squared Error). Some experiments suggest that increasing the number of cities can lead to better generalization of patterns and features, but the overall results indicate that these gains are limited and may not be relevant.

Acknowledgments

This thesis was written as part of my Master's degree in Mechanical Engineering at the Technical University of Munich at the Transporting Systems Engineering chair. First, I would like to express my gratitude to Univ.-Prof. Dr. Constantinos Antoniou and M. Sc. Cheng Lyu for allowing me to work on this project. They provided me with great counseling and relevant discussions, bringing up the level of this work.

I want to express my gratitude to Univ.-Prof. Dr. Larissa Driemeier for serving as my co-supervisor in this thesis and to the Polytechnic School of the University of São Paulo for allowing me to study at TUM as part of a double degree program.

I also want to thank my friends with whom I discussed this thesis and from whom I found good listeners and advisors. In special, Lui, my companion during endless debugging sessions; Luís, João Pedro, and Ariel, my roommates and friends whom I used as rubber ducks, explaining problems and bugs in detail until I could figure out the causes; and Bruno, for the diligent revisions he performed on my writings.

I am also very thankful to God for the strength, resilience, and courage during the writing of this thesis and for the entirety of my degree. To my parents, Ricardo and Camila, with whom I found support and love during this journey, this thesis is dedicated to you. I also thank all my siblings, José, Luís, Tomás, Sofia, Beatriz, and André, for their encouragement and help. I would also like to thank my uncle, aunt, and cousin, Hermann, Luciana, and Hermann, whose home was a second home for me during this year.

Finally, I thank everyone with whom I shared this part of my life, full of discoveries, a bit of hardship, and rich in personal growth.

Contents

List of Figures

List of Tables

List of Algorithms

Glossary

A

A3T-GCN - Attention Temporal Graph Convolutional Network	29, 48, 59
AE - Autoencoder	47, 53
AI - Artificial Intelligence	1
ARIMA - Autoregressive Integrated Moving Average	36, 51

B

BCE - Binary Cross Entropy	26, 27, 35

C

CNN - Convolutional Neural Network	13
ConvLSTM - Convolutional Long Short-Term Memory	13, 36, 37, 51
CTD - Common Tucker Decomposition	15

D

DD - Domain Discriminator	26, 30, 31, 47

G

GCN - Graph Convolutional Network	11, 13
GConvLSTM - Chebyshev Graph Convolutional Long Short-Term Memory	23–25, 36, 59
GLU - Gated Linear Unit	23
GRL - Gradient Reversal Layer	15, 16, 25, 26, 30, 31, 57
GRU - Gated Recurrent Unit	13

H

HA - Historical Average	36, 51

I

IQR - Interquartile Range	40, 41, 43–46, 49

L

LSTM - Long Short-Term Memory	11, 13, 24, 26

M

MAE - Mean Absolute Error	1, 5, 33–35, 39–46, 48–51, 57
MAPE - Mean Absolute Percentage Error	33, 34
MMD - Maximum Mean Discrepancy	15
MSE - Mean Squared Error	1, 5, 33, 35, 39–51, 53, 57
MSLE - Mean Squared Logarithm Error	34, 42, 43

N

NLP - Natural Language Processing	14

1. Introduction

1.1. Motivation

According to [1], a *smart city* can be defined as a well-coordinated system that integrates advanced technological infrastructure, relying on sophisticated data processing. The primary objectives of such integration are to enhance city governance efficiency, improve citizen satisfaction, foster business prosperity, and promote environmental sustainability. Within a *smart city*, the management of various individual systems that constitute the urban environment is not solely reliant on the data collected within the city; it also relies on different adaptable models that can learn and evolve to suit the specific needs and characteristics of the city. In this context, the development of traffic prediction models emerges as a pivotal component for establishing the foundational framework of *smart city* management.

Recent advances in deep learning, fed by the advent of Big Data, have led to significant advancements in prediction tasks related to traffic, such as crowd flow [2, 3], traffic flow [4, 5], public transit flow [6, 7], travel demands [8], and traffic speeds [9]. While formidable in their predictive power, these models come with a substantial data appetite. This data requirement poses a challenge for initiating new intelligent networks because meaningful inferences remain elusive despite the considerable investment needed to establish the sensor network without access to substantial data history [10]. This difficulty is known in the field as the "cold-start" problem [11], and has a higher burden on small and medium cities, as "they have relatively rare knowledge of their historical patterns" [10].

To address the aforementioned challenge, novel techniques rooted in transfer learning [11] have been introduced. These approaches enable training predictive traffic models for cities constrained by limited data by taking advantage of patterns observed in cities with abundant data resources. On the same paper, the authors provide an overview on transfer learning problems, and segregate the possible setups into five categories. The first one is composed by the cases where there is no labeled data on both domains, but the target task is related to the source one.

The second possible setup is called Transductive Transfer Learning, happens when there is labeled data only for the source domain. If we assume that the domains are the same, we would end up with Covariance Shift, and otherwise, if we assume that the domains are different, but the task is the same, we would be performing Domain Adaptation.

Finally, if we do have labeled target data, that's a case of Inductive Transfer Learning. If, additionally, there is labeled data on the source domain, we learn both tasks simultaneously, on what is called Multi-Task Learning. If there is no labeled data for the source domain, we end up with Self-taught Learning.

Our problem consists in creating a regression model capable of learning patterns from a data-rich source domain and transferring this knowledge to the target domain, assumed to be data-scarce. On this setup, the definition of Domain Adaptation is the one that makes the most sense, as we would be learning the same task over two different domains where the target domain lacks labeled data. This type of technique entails initializing the network in the source city and implementing fine-tuning to adapt the network to the unique characteristics of the target city.

Additionally, the data representing the dynamic progression of urban traffic is inherently complex, encompassing two spatial dimensions and one temporal dimension. This multifaceted data is often expressed as a graph structure, where distinct segments or areas of the city are depicted as nodes, interconnected by unweighted edges constituting the physical connections between neighbor nodes. Effectively processing and interpreting this data requires specialized approaches. GCN (Graph Convolutional Network) are generally employed for this purpose, adept at handling the intricacies of such graph-based representations.

The temporal aspect of the data is observed as a sequence of graphs that, generally, preserve their structure (i.e. the connectivity) while varying the node and edge attributes. These variations can be modeled with the help of recurrent networks, such as LSTM (Long Short-Term Memory).

Currently, state-of-the-art models focus on exploring the application of new deep learning frameworks, such as attention, and transformers, instead of leveraging diverse source data when generating domain-agnostic features in transfer learning. Some works consider the possibility of using multiple source cities in the domain adaptation process [12, 13, 14]. Still, as of now, few paper delves into the impact diverse source domains can have on accuracy.

1.2. Research Questions

The main motivation of this thesis is to build a model capable of predicting the next state of a given traffic system. To reach this goal, the following objectives were drawn:

- Analyze current state-of-the-art models and identify cells and architectures that could be used when building a novel model;
- Based on the results of the first objective, propose a novel model to be built, which should follow these requirements:
  - capable of learning from multiple cities at the same time;
  - capable of intra-city learning and

    - capable of considering external features (such as weather data, POI (Point of Interest) locations, relative date features).

- Analyze how impactful each feature derived from a requirement is to the model.

Concurrently, the following research questions were raised:

Q.1 Is it possible to encompass more than two cities as sources in a transfer learning process?

Q.2 What's the impact of the number of source cities on the model's accuracy?

Q.3 Is there a limit on the number of source cities?

1.3. Contribution

With this work, we aim to contribute to the traffic prediction and transfer learning fields by proposing a novel model composition which explores the idea of gathering a big and diverse source domain to acquire knowledge from. The understanding of the impacts that this diversity exercises on the model's performance is key to the development of other models capable of absorbing even more knowledge.

Furthermore, we also aim to understand how the size of the target dataset impacts the model's ability to generalize and absorb a transferred knowledge. Through many experiments, we aspire to test and demonstrate the effectiveness of all model components, as well as the transfer mechanisms.

1.4. Outline

This work is organized as follows Chapter 2 introduces the literature review that was performed to further understand the problem and provides a comprehensive explanation of commonplace concepts of the field. Chapter 3 proposes a methodological framework for the entire work, including data acquisition, pre-processing, model building, and testing setup. Chapter 4 analyzes the results of the proposed tests and comparisons to proposed baselines. Chapter 5 uses the results to answer and discuss the research questions raised in Chapter 1. Chapter 6 concludes the thesis and discusses the main directions for future research in the field.

All developed code and assets (such as models and notebooks) are available on `https://github.com/jolenscki/master-thesis`.

2. Literature Review

This chapter presents a literature review that we conducted to further the traffic forecasting field and its state-of-the-art.

2.1. Traffic Forecasting

Traffic Forecasting is a long-lasting field of study in Traffic Engineering, conceived in the 1950s [15, 16]. Initially centered on traffic simulation, the area observed a significant upward trend in recent years. By its very nature, a traffic network constitutes a vast and complex system where events occurring at various junctures within the road grid can exert profound influence over the entire traffic flow. These inherent complexities render it an ideal subject for examination by cutting-edge machine-learning algorithms, such as LSTM, GRU (Gated Recurrent Unit), and CNN (Convolutional Neural Network).

Traffic network problems inherently comprise two domains: the temporal and spatial domains. In the early stages of deep learning model development for the field of traffic, a natural division emerged, allocating separate components to address each of these domains. For example, CNN has conventionally been harnessed primarily for spatial feature extraction, which involves identifying elements' physical locations or arrangements within a given dataset. Conversely, RNN (Recurrent Neural Network) has specialized in temporal feature extraction, focusing on discerning patterns that evolve or change over time.

The pioneering work of [17] marked one of the earliest instances of employing CNNs for traffic prediction tasks. Subsequently, this methodology proliferated across various model frameworks and underwent integration with other architectural paradigms. An illustrative example of this evolutionary trajectory emerges from the study conducted by [18], which employed a multiple GCN framework. In this network type, the input to the convolutional layers consists of the city's graph representation.

Similarly, [19] was among the early proponents of employing LSTM cells for short-term traffic prediction. This approach gained substantial traction, integrating cells into various models, as exemplified by the ConvLSTM (Convolutional Long Short-Term Memory) module introduced by [20]. In this architecture, spatial features were extracted at each time frame and subsequently fed into an LSTM chain. This innovative approach allowed for the extraction of spatial and temporal features concurrently.

As a complex ST problem, traffic forecasting offers a range of strategies, spanning from problem formulation to data structuring, encompassing data type selection. Regarding data representation, many authors [21, 12, 22] choose to apply a grid in the city and treat each

1-by-1 region autonomously, computing variables (inflow and outflow, for instance) inside these boundaries. In this approach, the instantaneous snapshot of the city could be compared to an image in the context of image classification algorithms, with each pixel being equivalent to a region.

An alternative to this structure is transforming the raw data, typically provided as a matrix or tensor, into a graph-based representation, with each geographical region converted into a distinct node and an adjacency matrix defining the connections between neighboring regions [23, 24, 18, 25, 26, 27, 13, 14, 28]. This method can better capture the network's complexity and regions' nuanced connectivity. In contrast to the grid-based approach, where regions may share a border without necessarily sharing a connecting road, the graph representation effectively accounts for such subtleties.

A third strategy, applied in the works of [29, 30], involves defining individual road segments as graph nodes. This approach offers significantly greater detail and precision in modeling traffic data as the data sources are condensed in a relatively small area. As a drawback, it also requires the installation of many more sensors to produce the data.

2.2. Transfer Learning

Transfer learning techniques [11] were introduced as valuable tools for addressing problems where existing knowledge or expertise from one domain could be employed to enhance learning and performance in another domain. These techniques prove incredibly beneficial when the latter domain suffers from a shortage of data. This can be applied in NLP (Natural Language Processing) problems such as sentiment and document classification, where labeling data is a very demanding task. Furthermore, the computer vision field also benefited greatly from applying these approaches, as they are extensively used for image classification.

As many cities started to prepare themselves to transition into *smart cities*, they stumbled upon the “cold start” problem. This problem refers to the lack of data that a city faces after installing the sensor network and before acquiring enough data to justify deep learning models, and it is not unique to *smart cities*' context but resonates with the broader field of machine learning [31]. In such a situation, managers must wait a certain amount of time until they can make reasonable predictions with deep learning models despite all investments made in traffic prediction. In these cases, based on the assumption that despite being different, some cities can share a common behavior framework and have similar data distributions. Transfer learning favors acquiring knowledge from cities with abundant data and using this knowledge to understand and predict cities with scarce data.

Furthermore, Transfer Learning techniques are also suitable for intra-city transfer, i.e., transferring the learned patterns from a domain (for instance, bike sharing flow) to another in the same city (for example, pedestrian flow). This can observed in the work of [22], in which the author used a taxi trip dataset to learn about bike sharing in the same city.

Generally, a transfer learning algorithm consists of three parts: feature extraction, in which ST features are obtained through various ways; domain adaptation, in which the knowledge is transferred; and a predictor, responsible for making a prediction on the next step based on the extracted features obtained from the aforementioned parts.

The feature extraction step is also present in deep learning approaches to the traffic forecasting problem and it can be achieved by many different architectures, as discussed in Section 2.1. On the other hand, the domain adaptation step is mainly present in the transfer learning networks and aims to extract transferable latent features between the domains. With it, one seeks to learn domain-invariant knowledge about the problem.

2.3. Domain Adaptation

On the efforts of domain adaptation, [22] proposes using convolutional layers parallelly for both source and target features. These convolutional layers are interconnected by calculating the MMD (Maximum Mean Discrepancy) between layers to form a transfer loss, which is then added to the overall model loss. MMD is a statistical metric that quantifies the dissimilarity between two probability distributions [32]. By using this metric as a loss function, the authors aimed to make the feature representation of different domains more similar. In like manner, [24] also proposes the use of MMD to minimize the distance between features of different features and, in addition, the use of binary cross-entropy on the classification of the edge types, as the authors use multi-view graphs.

Despite working in a different field (network traffic prediction), [33] uses a similar architecture to [22], with the substitution of the MMD cells for CTD (Common Tucker Decomposition) ones. With a similar loss calculation structure, the CTD cells aim to measure the domains' discrepancy. [23] explores a different approach. In their work, the authors propose using adversarial training for transfer learning. This implies the design of a discriminator and a predictor in the network to generate a transfer loss.

A different, more typical approach is used in [14], as the authors employ the parameter-sharing technique: pre-training on the source domain and fine-tuning on the target domain. In this technique, the parameters of the fine-tuning stage are initialized with the trained parameters from the pre-training phase. This adaptation is only possible when we assume that the difference between distinct domains is insignificant and that both domains may share common features and patterns.

In the same paper, [14] also implements another knowledge transfer technique, the GRL (Gradient Reversal Layer), proposed initially by [34]. This architecture features an identity forward pass and reverses the gradient signal during the backward pass. Considering that different cities possess unique spatial structures and compositions, the GRL is designed to mitigate these discrepancies by fostering an adversarial training environment. This is achieved by reversing the gradient, which confuses the model during training and is coupled with a domain classifier. The domain classifier's task is to differentiate source and target domains. At the same time, the model, through the influence of the GRL, learns to generate domain-invariant features, thus enhancing its ability to generalize across different city datasets. This approach is convenient in scenarios where the goal is to adapt a model trained on one city's data (source domain) to perform accurately on data from another city (target domain) despite the inherent differences in their spatial characteristics.

3. Methodology

In this chapter, we build a complete picture of the Methodology applied during the subject research. We briefly analyze the dataset used and describe all components of the implemented model and the training script used to generate the optimal weights for these components.

3.1. Data Analysis and Exploration

The dataset selected for sourcing the prediction model was part of the NeurIPS2021 Traffic4cast competition [35]. This dataset consists of 360 days of data from 8 different cities with similar sizes derived from trajectories of a fleet of probe vehicles. The original data has 180 days from 2019 and another 180 days from 2020, as one of the questions of the challenge was to asses how the COVID pandemic affected traffic in different cities. As this represents a shift in the temporal distribution of the data, we choose not to use the 2020 half of it, intending to have an assumed temporal invariant distribution. Table 1 disposes of all cities available on the dataset and the number of data points each city contains.

City	**# of days**	**# snapshots per day**
Antwerp	180	240
Bangkok	180	240
Barcelona	180	240
Berlin	180	240
Chicago	180	240
Istanbul	180	240
Melbourne	180	240
Moscow	180	240

Table 1 List of the cities available on the dataset

For a given time snapshot, the data of one city can be represented by a tensor of size $(495, 436, 8)$, where 495×436 represents the city grid, and 8 stands for the channels, or pieces of information, per cell. These channels contain information on the volume and mean speed of the probe cars heading in the four diagonal directions. All data was normalized and discretized in the `uint8` range, meaning values are contained in the $[0, 255]$ range.

To better understand the data, distribution, and characteristics, the following analysis is conducted on the 5-minute snapshot that started at 12:00 on 9 January 2019 in Melbourne. Figures 1 and 2 show the distribution of values for both the speed (odd channels) and the volume (even channels). It's clear that while the volume has a more even distribution for non-zero values, there remains a massive bias for the zero values, as they represent more than 99% of the data points. This indicates that most of our data comprises zeros and that activity should be treated as rare.

Furthermore, Figure 3 confirms this theory, as it can be seen that most of the data for the speed channels is zero, and the volume, despite being better distributed, is also defined by a majority of zeros. Table 2 shows the statistics for each channel and reinforces the proposed thesis.

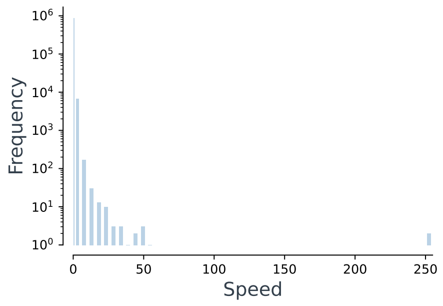

Figure 1 Histogram for data distribution for the speed. Note the first (very thin) bin with the null values.

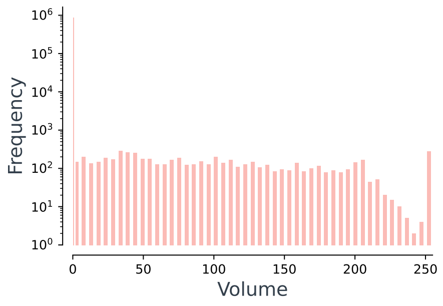

Figure 2 Histogram for data distribution for the volume. Note the first (very thin) bin with the null values.

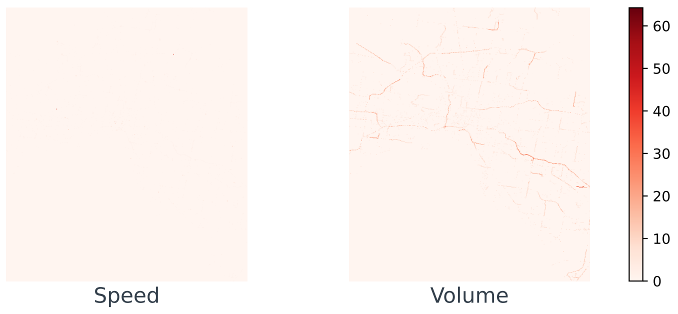

Figure 3 Heatmap of the snapshot for the (a) Speed; and (b) Volume.

3.1.1. Data processing

Some other data processing transformations were processed besides disposing of the 2020 half of the original data. A significant part of the motivation for this processing step is to reduce the input data's size (or shape) to make the models trainable in a reasonable time, given the limited computational resources available. The first transformation implemented was the collapse of the even (volume) and odd (speed) channels by taking the mean of the data in every channel. Therefore, we were left with two channels: the average volume and average speed of the cars in the particular region.

Furthermore, to allow efficient development of the model's components, only the central

Channel	Mean	Median	Standard Deviation	Non-zero elements
0 (volume)	0.0216	0.0000	0.2861	1.08%
2 (volume)	0.0141	0.0000	0.2794	0.84%
4 (volume)	0.0133	0.0000	0.6236	0.72%
6 (volume)	0.0118	0.0000	0.5957	0.62%
1 (speed)	0.6758	0.0000	9.5058	0.73%
3 (speed)	0.8655	0.0000	11.3655	0.82%
5 (speed)	0.7419	0.0000	10.5863	0.71%
7 (speed)	0.6149	0.0000	9.8281	0.60%

Table 2 Statistics for each channel in the specific snapshot

square of size 50×50 was employed for the preliminary tests and experiments, including those used to tune the model's parameters. This may seem limiting at first glance, as it reduces the spatial coverage and may exclude potentially significant peripheral data. Nonetheless, this strategy is practically beneficial as it simplifies computational demands while capturing a substantial portion of traffic behaviors. The selected central area includes key urban sections characterized by informative traffic dynamics, which will help scale the model to encompass the entire urban layout.

Finally, we normalize the data again, form $[0, 255]$ to $[0, 1]$, as dealing with float points in this interval can be considerably favorable.

3.2. Model Outline

The proposed model consists of two modules: a Feature Extraction Network, modeled as the encoder of an autoencoder and explained in Section 3.3, and a Prediction Network, thoughtfully dissected in Section 3.5. Furthermore, Section 3.4 is reserved for the exposition of the domain adaptation techniques employed through the model and during training to enable knowledge transfer between the domains. Section 3.6 summarizes with figures and pseudocode scripts how the model's training was done. This Section will explain the mathematical and formal basis for the model and the overall architecture to be built.

3.2.1. Task

The field of Spatio-Temporal prediction is vast, allowing researchers to try different approaches to problems that may seem the same, and this can be observed from the bibliography presented in the bibliographic revision. Therefore, it's essential to clearly define the problem to be solved and rigorously define the available information.

Definition 1. *A city C is divided into a grid map of shape $W_C \times H_C$. Each partition $r_{i,j}$, with $1 \leq i \leq W_C$ and $1 \leq j \leq H_C$, is referred to as a region of C. The set containing all regions of the city is defined $R_C = \{r_{1,1}, ..., r_{W_C,H_C}\}$.*

Definition 2. *The time range of available data of a city is divided into T_C intervals of equal size: $t = [1, ..., t_C]$.*

Definition 3. *For each region $r_{i,j}$, N_{ch} data channels are available. These channels are the same for every city.*

By combining Definitions 1, 2, and 3 we can visualize the the 4D tensor of shape (W_C, H_C, N_{ch}, T_C). Figure 4 shows a slice of this tensor. Each color represents a different channel, and each square of four colors represents a point in the 2D city grid. The stacked layers represent the time dimension.

Definition 4. *A 4D tensor defines the data of a city C:*

$$\mathcal{X}_C = \{x_{r,t}^{ch} | r \in R_C, t \in T_c, ch \in N_{ch}\} \tag{3.1}$$

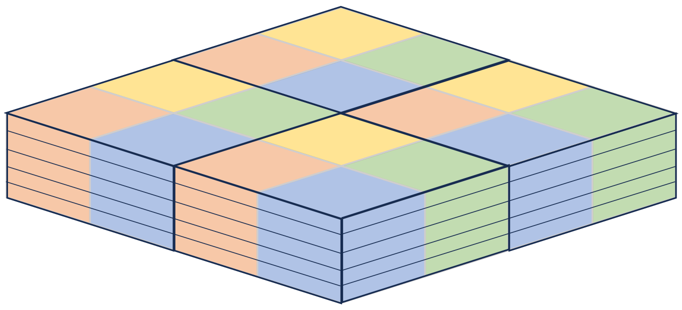

Figure 4 Visualization of the data (as a tensor).

Definition 5. *Connectivity in the city grid, denoted as $\mathcal{E}_C$, is defined by the set of edges that establish the relationship between adjacent regions in C. Each edge $e_{k,l}$ in $\mathcal{E}_C$ connects two regions $r_{i,j}$ and $r_{m,n}$, where i and m are indices within the width of the grid W_C and j and n are indices within the height of the grid H_C. An edge exists if the regions it connects are* *adjacent horizontally, vertically, or diagonally. Formally, the connectivity is represented as:*

$$\mathcal{E}_C = \{e_{k,l} | e_{k,l} \text{ connects } r_{i,j} \text{ and } r_{m,n}, \text{ such that } |i-m| \leq 1 \text{ and } |j-n| \leq 1,$$
$$\text{for all } i,j,m,n \text{ that satisfy } 1 \leq i,m \leq W_C \text{ and } 1 \leq j,n \leq H_C\}$$

Note that the grid connectivity is associated with the actual underlying road network, with edges representing the physical connection that exists between the regions.

Definition 6. *A graph representation of a city C, denoted as G_C, is defined as a tuple $(V_C, \mathcal{E}_C, \mathcal{X}_C)$, where V_C is the set of nodes corresponding to the regions of C, $\mathcal{E}_C$ is the set of edges representing connectivity between regions, and $\mathcal{X}_C$ is the node feature matrix, containing the feature (or channel) values of each node in V_C.*

Putting all previous definitions together, we define the adjacency matrix:

Definition 7. *The adjacency matrix for a city C, denoted as $\mathbf{A}_C$, is a matrix that represents the connectivity between the nodes of C as defined in $\mathcal{E}_C$ from Definition 5. The adjacency matrix is a square matrix of size $|V_C| \times |V_C|$, where $|V_C|$ is the number of nodes in the graph representation of the city. Each element a_{ij} of the matrix $\mathbf{A}_C$ is defined as follows:*

$$a_{ij} = \begin{cases} 1, & \text{if there exists an edge } e_{ij} \in \mathcal{E}_C \text{ connecting nodes } v_i \text{ and } v_j \\ 0, & \text{otherwise} \end{cases}$$

This matrix is symmetrical, as the presence of an edge e_{ij} implies bidirectional connectivity between the nodes v_i and v_j.

With these definitions, it's possible then to define Problem 1:

Problem 1. *Given a data-scarce target city, C_T, and a set of n data-rich source cities $\{C_{S1}, ..., C_{Sn}\}$, the problem proposed is to predict the value of the target city's data at $t_T + 1$ with the historical data of the target city itself to that point and of the source cities:*

$$\min_{\theta} \mathcal{L}(\tilde{\mathcal{X}}_{T,t_T+1}, \mathcal{X}_{T,t_T+1}) \tag{3.2}$$

where

$$\tilde{\mathcal{X}}_{T,t_T+1} = \theta(\mathcal{X}_{T,1:t_T}, \{\mathcal{X}_{S1}, ..., \mathcal{X}_{Sn}\}) \tag{3.3}$$

Note that $\mathcal{L}$ is the error criterion, which may be tuned depending on our dataset and model specifications. Note also that $t_{Sk} \gg t_T \forall k = 1, ..., n$, indicating the target city's scarcity and the sources' richness.

3.2.2. Proposed Architecture

As explained at the beginning of this Section, the model comprises a Feature Extractor block modeled after the encoder of an autoencoder and a Predictor block. Figure 5 outlines the proposed architecture. Note that the individual modules will be developed and explained in their sections. The architecture adopted in this research draws upon the works of [33, 22] as they proposed similar divisions in their models to transfer knowledge. Compared to their approach, we suggest using a STGAE (Spatio-Temporal Graph Autoencoder) to train the feature extractor and an adjacency matrix to represent connectivity between the regions. Furthermore, the training part of the presented model is influenced by the work of [14], as we use similar domain adaptation techniques but delve into the influences that diversity on the source domain exerts on the model's accuracy.

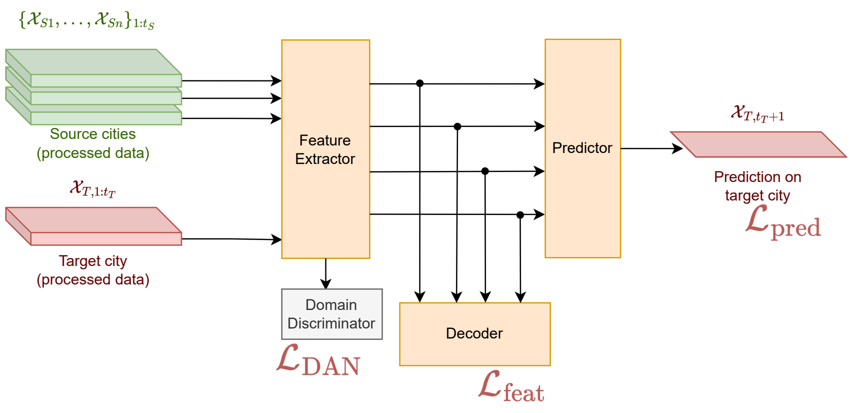
$\{\mathcal{X}_{S1}, \ldots, \mathcal{X}_{Sn}\}_{1:t_S}$
Source cities (processed data)
$\mathcal{X}_{T,1:t_T}$
Target city (processed data)
Feature Extractor
Domain Discriminator
$\mathcal{L}_{\text{DAN}}$
Decoder
$\mathcal{L}_{\text{feat}}$
Predictor
$\mathcal{X}_{T,t_T+1}$
Prediction on target city
$\mathcal{L}_{\text{pred}}$

Figure 5 Simplified version of the proposed model.

3.3. Feature Extraction Network

As the first layer of the model, the Feature Extraction Network receives an input of tensors from one or more source cities and the target city and tries to extract, from these tensors, ST features must be extracted to be used to train the Prediction Network.

3.3.1. Autoencoder

Selecting hyperparameters and fine-tuning an extractor are challenging tasks when constructing a model, as it's difficult to observe causality between the change of a parameter and the change of the output due to the highly non-linear characteristics of these modules. As a possible solution to these problems, the use of autoencoders for the feature extraction task has been proposed by [36], and it's, as of today, a well-established paradigm in the ST field. By conceptualizing the feature extraction process through the lens of an encoder, it becomes easier to verify its quality by constructing a corresponding decoder. As the encoder maps the input data from its original vectorial space to a latent space, the decoder pursues the contrary operation, returning the data from the latent space to the original one. In an ideal scenario, a well-trained autoencoder will reconstruct the original data perfectly, guaranteeing the quality of the features extracted by the encoder.

More recently, [37] implemented a STAE (Spatio-Temporal Autoencoder) by coupling GLU (Gated Linear Unit) layers for time convolution and Chebyshev convolution layers for spatial convolution. By interpolating two temporal layers by a spatial one, the authors extracted both spatial and temporal features. Additionally, using Chebyshev filters of relatively large sizes $(K = 6)$, the proposed autoencoder could correctly derive features on both local and global scales.

In another paper, [38] proposes a generic framework for STGAE with symmetric encoder-decoder architectures. The encoder finds a latent graph representation by applying graph convolutions, temporal downsampling layers, and activation functions. The decoder mirrors this behavior but uses temporal upsampling layers between the convolutions and activation functions.

Figure 6 illustrates the architecture of the STGAE utilized in our feature extraction framework. The encoder predominantly comprises a GConvLSTM (Chebyshev Graph Convolutional Long Short-Term Memory) block for extracting spatio-temporal features. It is followed sequentially by an activation function, batch normalization, a regularization dropout layer, and a linear transformation layer. Similarly, the decoder incorporates analogous components, adding a Sigmoid activation function preceding the output. This configuration leverages the data normalization previously applied, wherein the value range of all channels was linearly transformed from $[0, 255]$ to a unit interval $[0, 1]$.

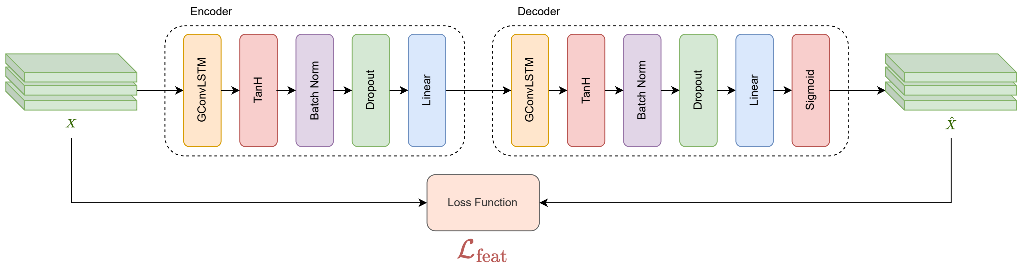

Figure 6 Diagram representing the autoencoder as a combination of an encoder and a decoder.

The GConvLSTM cell, as implemented by [39], and initially proposed by [40], is parameterized by the number of input channels N_{in}, the number of output channels N_{out}, and the size of the Chebyshev polynomial filter K. The cell executes graph-based convolutions on the input tensor x, with the knowledge of the graph edge's descriptor tensor `edge_index`, to yield the hidden state h and the cell state c. These states are then propagated through the sequence for subsequent iterations, defined by the k discrete temporal segments of which the input x is composed. This enables the model to capture and encode the temporal dynamics of the data.

$$\text{GConvLSTM} : \mathbb{R}^{D_1 \times D_2 \times \ldots \times D_N \times N_{\text{in}}} \to \mathbb{R}^{D_1 \times D_2 \times \ldots \times D_N \times N_{\text{out}}} \tag{3.4}$$

Furthermore, as implemented in this layer, the order of the Chebyshev filter plays a pivotal role, as it defines the range of neighborhood aggregation. Specifically, it dictates how and how many local neighborhoods are expanded around each node during the convolutional process. This, in turn, influences the gradient computation during backpropagation, affecting both the receptive field and the capacity of the model to capture and integrate multi-hop relational information.

In a regular LSTM implementation, all internal variables are calculated based on the sigmoid of combinations of fully connected layers as highlighted in the equations:

$$\begin{aligned}
i_t &= \sigma\left(\boxed{W_{xi}x_t} + \boxed{W_{hi}h_{t-1}} + w_{ci} \odot c_{t-1} + b_i\right), \\
f_t &= \sigma\left(\boxed{W_{xf}x_t} + \boxed{W_{hf}h_{t-1}} + w_{cf} \odot c_{t-1} + b_f\right), \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tanh\left(\boxed{W_{xc}x_t} + \boxed{W_{hc}h_{t-1}} + b_c\right), \\
o_t &= \sigma\left(\boxed{W_{xo}x_t} + \boxed{W_{ho}h_{t-1}} + w_{co} \odot c_t + b_o\right), \\
h_t &= o \odot \tanh\left(c_t\right),
\end{aligned}$$

Generalizing the LSTM, a model developed for time-series forecasting, for graph inputs requires the adjustment of these operations for something that can handle graph-data input. For the GConvLSTM, the authors implemented the graph convolution operator $*_{\mathcal{G}}$ proposed by [41], in which a graph signal $x \in \mathbb{R}^n$ with n nodes is filtered by a non-parametric kernel g_θ composed of vectors of Fourier coefficients.

$$y = g_\theta *_{\mathcal{G}} x$$

On this implementation, $*_{\mathcal{G}}$ is modeled with the normalized graph Laplacian decomposition $L = U\Lambda U^T$, which would imply a model's complexity of $\mathcal{O}(n^2)$. To make its use more feasible, the authors propose a truncated expansion of g_θ using Chebyshev polynomials T_k and truncated laplacian $\tilde{L}$. This reduces the complexity to $\mathcal{O}(|\epsilon|n)$, with ϵ number of edges of the graph.

$$y = g_\theta *_{\mathcal{G}} x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x$$

Finally, it's possible to define the GConvLSTM model.

$$\begin{aligned}
i_t &= \sigma\left(W_{xi} *_{\mathcal{G}} x_t + W_{hi} *_{\mathcal{G}} h_{t-1} + w_{ci} \odot c_{t-1} + b_i\right), \\
f_t &= \sigma\left(W_{xf} *_{\mathcal{G}} x_t + W_{hf} *_{\mathcal{G}} h_{t-1} + w_{cf} \odot c_{t-1} + b_f\right), \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tanh\left(W_{xc} *_{\mathcal{G}} x_t + W_{hc} *_{\mathcal{G}} h_{t-1} + b_c\right), \\
o_t &= \sigma\left(W_{xo} *_{\mathcal{G}} x_t + W_{ho} *_{\mathcal{G}} h_{t-1} + w_{co} \odot c_t + b_o\right), \\
h_t &= o \odot \tanh\left(c_t\right),
\end{aligned} \tag{3.5}$$

3.3.2. Fine Tuning

The autoencoder, the most computationally demanding part of the entire model, owes much of its complexity to the GConvLSTM layers in both the encoder and decoder. These layers execute graph convolution operations, which can become computationally intensive for large graphs.

We suggest a pragmatic two-step training approach for the feature extractor to handle the computational demand more effectively. Initially, we train the autoencoder using the available source data, which allows us to establish a solid initial data representation. Subsequently, we fine-tune this representation with the target data, adapting it to the specific characteristics of the target domain.

Moreover, this method results in a fixed feature extractor not tightly coupled to the overall model. This flexibility is particularly beneficial when considering the parameter definition of the latter parts of the model, meaning that we can then focus on training the Prediction Network without feeling the effects that the choice of the feature extractor's parameter has on the results.

3.4. Domain Adaptation

As one of the main challenges of a transfer learning task, domain adaptation is the process of adapting a model trained on one or more source domains (where abundant data is available) to perform well on a different but related target domain (where data is limited or has other distribution characteristics). Various approaches can be taken to tackle this problem in the different contexts that appear during the development of the model.

We suggest using two domain adaptors for this model: parameter sharing and a GRL coupled with a Domain Discriminator. The parameter sharing comprises the pre-training of the model using the source domain, and a fine-tuning of this model using the target domain, but starting the model's weights with the final values obtained with the source data.

The GRL, when coupled with a Domain Discriminator, on the other hand, aims to help the feature extractor to produce domain-invariant features, which will then be extremely useful for transferring knowledge between different domains by making the encoder less sensitive to the specific characteristics of the source domain, thereby enhancing its ability to perform well on the target domain. It's to be placed just after the encoder block of the autoencoder and before the decoder.

This joint effort between the GRL and the DD (Domain Discriminator) is also known as adversarial training strategy. While we encode and reconstruct the original data with the autoencoder (which generates the reconstruction loss), the domain discriminator or classifier tries to distinguish source and target domains. Since the GRL reverses the gradient sign and scales it during backpropagation, it encourages the model to generate features that try to "fool" the domain classifier, which raises the loss (as we expect from a generalization technique) but enforces the encoder to generate features that are domain-agnostic.

In the Domain Discriminator block, a sequence of operations is designed to process the graph features effectively. Initially, a Top-K Pooling operation is employed to reduce the node count over all the graphs, selecting the K_{pool} nodes with the most significant features from the last temporal segment output of the encoder. This process is essential for maintaining a consistent input shape for subsequent layers, ensuring the discriminator can operate uniformly on graphs from varying domains. After pooling, the features undergo batch normalization and dropout. The following linear layer projects the normalized and subsampled features onto a latent space, which a sigmoid function then squeezes into the $[0, 1]$ interval.

This operation translates the latent representations into probabilities, indicating the likelihood of the graph belonging to the target or source domain. These probabilities form the basis for computing the Domain Adversarial Loss with the BCE (Binary Cross Entropy) Loss function. Figure 7 illustrates the placement of the GRL and the Domain Discriminator within the overall model structure.

Given the output of the encoder block $H \in \mathbb{R}^{k \times N \times D_{lin}}$, with k being the number of discrete temporal segments of which the input was composed, N being the batched number of nodes of the input, and D_{lin} the tunable dimension of the output of the linear layer of the encoder, we first select the last temporal segment from H. This choice is predicated on the understanding that the final snapshot encapsulates the most refined representation of the input graph, having undergone the full extent of recurrent processing through the LSTM layers.

The next step in the pipeline is to apply the Top-K Pool to the graph. This is crucial in harmonizing node dimensionality across graphs from varied domains, which inherently may have different number of nodes. This will standardize the input size for the domain discriminator, allowing it to consistently and fairly assess the domain characteristics regardless of the original graph size. By retaining only the top K_{pool} most significant nodes as per their learned representation in the encoder, Top-K Pooling concentrates on the most informative parts of

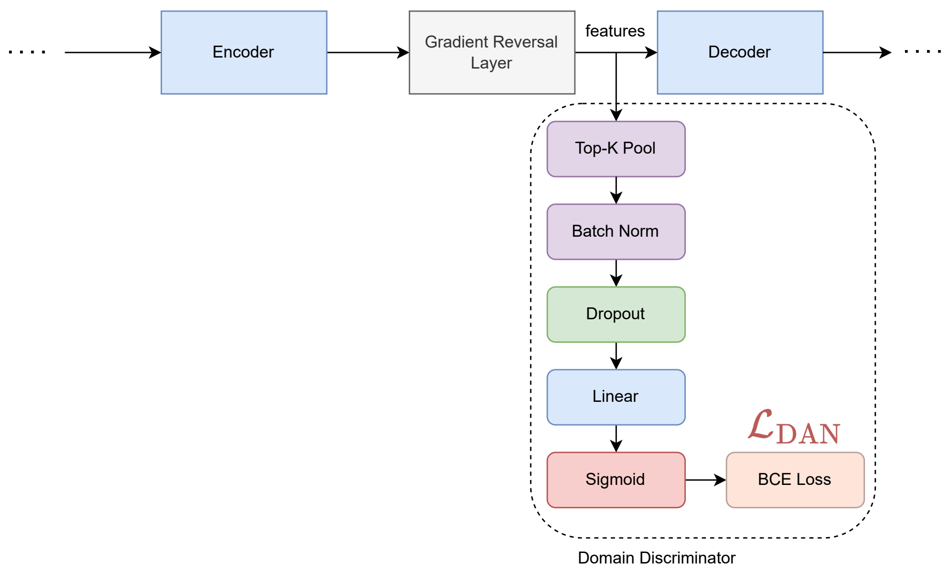

Figure 7 Diagram representing the architecture for the Domain Discriminator module.

the graphs, ensuring that the most critical structural and feature information is preserved for accurate domain discrimination.

$$\text{Top-K Pool} : \mathbb{R}^{N \times D_{lin}} \to \mathbb{R}^{K_{\text{pool}} \times D_{lin}} \tag{3.6}$$

Following the dimensionality reduction via Top-K Pooling, the processed graph features are channeled through a linear layer, which projects the features onto a scalar. The subsequent application of the sigmoid function maps these projections to a $[0, 1]$ range, with the output values near 0 or 1 interpreted as the probability of the graph belonging to the target or source domain, respectively. These probabilities are then used to determine the Domain Adversarial Loss using the BCE function - which will be presented and explained in Section 3.7.10.

The general loss for the autoencoder training will then be defined accordingly to Equation 3.7, with λ being a tunable regularization parameter for balancing both the reconstruction loss $\mathcal{L}_{\text{feat}}$ and this adversarial loss $\mathcal{L}_{\text{DAN}}$.

$$\mathcal{L}_{\text{AE}} = \mathcal{L}_{\text{feat}} + \lambda \cdot \mathcal{L}_{\text{DAN}} \tag{3.7}$$

3.5. Prediction Network

The second and final part of the model is the Prediction Network, which is responsible for making the next step prediction on the system state based on the features generated by the feature extractor and contextual timestamp features (time-of-the-day and day-of-the-week).

Given that the autoencoder developed is sufficiently good and can recreate the original input tensor x with high accuracy, it's possible to consider that the encoder has appropriately learned how to extract the relevant features from the input. Based on that assumption, we extract the encoder from the trained autoencoder and use it as the feature extractor. Note that for feature extraction, we would only require the encoder block of the autoencoder, as the decoder and the domain discriminator blocks are only helpful tools for training the encoder.

Figure 8 exposes the proposed architecture for the predictor block. It comprises an attention graph convolution block, an activation function, a linear layer, and a sigmoid normalization.

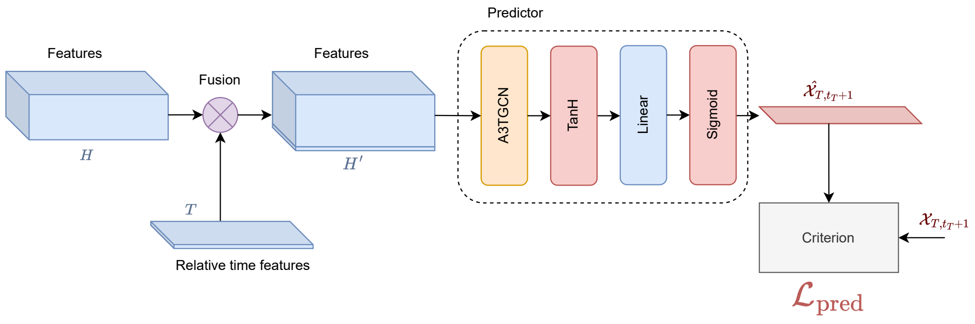

Figure 8 Diagram representing the architecture for the Predictor network.

The tensor H is the aforementioned output of the encoder block, while the tensor T is the tensor that encapsulates periodic relative time features within our dataset. These features are derived from a sine-cosine transformation to exploit the inherent cycles in traffic flow, such as the daily rush hour peaks or the variation in traffic between weekdays and weekends. For the hour-of-the-day, we calculate:

$$hour\_sin = \sin\left(2\pi \cdot \frac{current\_hour}{hours\_in\_day}\right), \quad hour\_cos = \cos\left(2\pi \cdot \frac{current\_hour}{hours\_in\_day}\right) \tag{3.8}$$

And for the day-of-the-week, we calculate:

$$day\_sin = \sin\left(2\pi \cdot \frac{current\_weekday}{days\_in\_week}\right), \quad day\_cos = \cos\left(2\pi \cdot \frac{current\_weekday}{days\_in\_week}\right) \tag{3.9}$$

Concatenating these results, we create the tensor $T = [hour\_sin, hour\_cos, day\_sin, day\_cos]$, providing a continuous and differentiable representation of time. This tensor is fused with the regular ST feature tensor, generating H'. For the fusion process, the tensor T was extended (by repetition) to have all but the last dimension match the shape of H. The fusion technique applied to join them was a simple concatenation.

The leading actor of the Predictor block is the A3T-GCN (Attention Temporal Graph Convolutional Network) cell, proposed by [42] and implemented in the *Pytorch Geometric Temporal* package [39]. It comprises two parts: first, a temporal convolution layer will generate N_t hidden states h for an input of N_t temporal snapshots. These hidden states will then be the input of an attention model that will determine the context vector capable of understanding global variation trends, with which we can predict the system's future steps.

3.6. Model Training

After introducing all components of the model, it's opportune to explain, in detail, how exactly these components are supposed to be trained. Figure 9 shows the proposed training script for the whole model.

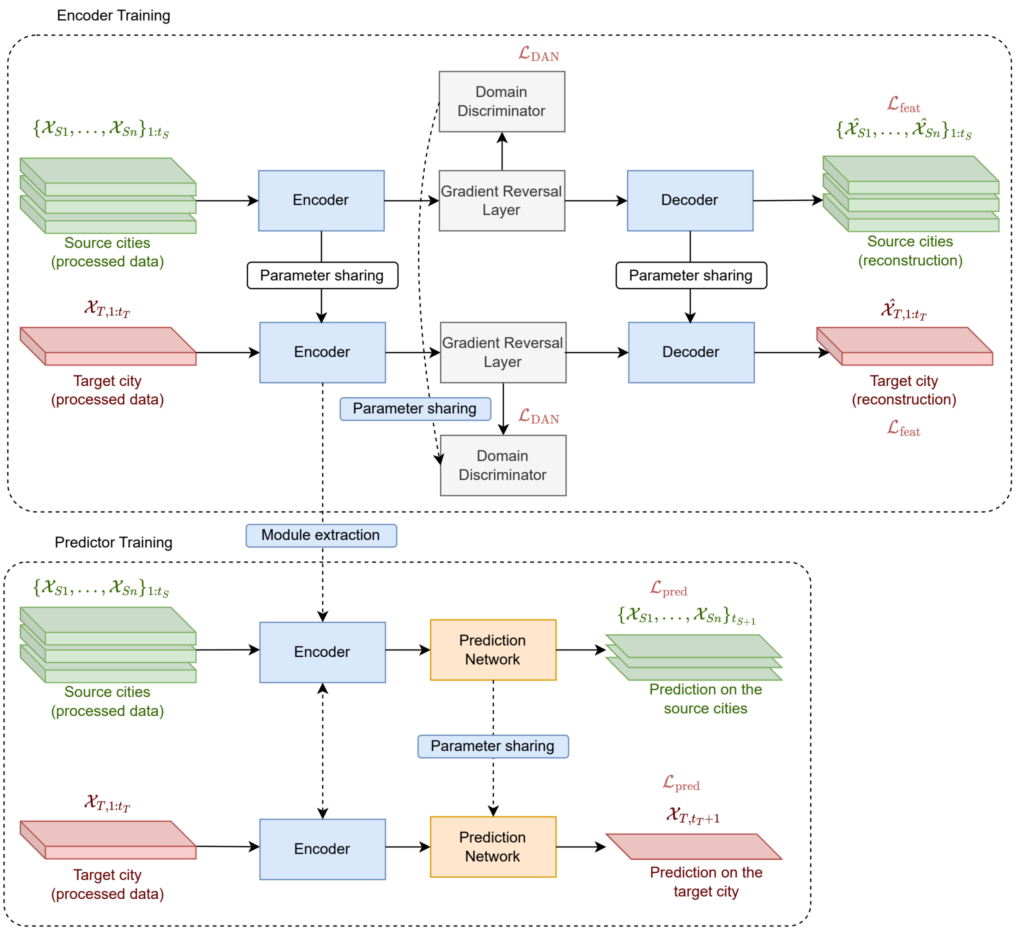
Encoder Training
$\mathcal{L}_{\text{DAN}}$
Domain Discriminator
$\{\mathcal{X}_{S1},\ldots,\mathcal{X}_{Sn}\}_{1:t_S}$
Source cities (processed data)
Encoder
Gradient Reversal Layer
Decoder
$\mathcal{L}_{\text{feat}}$
$\{\hat{\mathcal{X}}_{S1},\ldots,\hat{\mathcal{X}}_{Sn}\}_{1:t_S}$
Source cities (reconstruction)
Parameter sharing
Parameter sharing
$\mathcal{X}_{T,1:t_T}$
Target city (processed data)
Encoder
Gradient Reversal Layer
Decoder
$\hat{\mathcal{X}}_{T,1:t_T}$
Target city (reconstruction)
$\mathcal{L}_{\text{feat}}$
Parameter sharing
$\mathcal{L}_{\text{DAN}}$
Domain Discriminator
Module extraction
Predictor Training
$\{\mathcal{X}_{S1},\ldots,\mathcal{X}_{Sn}\}_{1:t_S}$
Source cities (processed data)
Encoder
Prediction Network
$\mathcal{L}_{\text{pred}}$
$\{\mathcal{X}_{S1},\ldots,\mathcal{X}_{Sn}\}_{t_{S+1}}$
Prediction on the source cities
Parameter sharing
$\mathcal{L}_{\text{pred}}$
$\mathcal{X}_{T,1:t_T}$
Target city (processed data)
Encoder
Prediction Network
$\mathcal{X}_{T,t_T+1}$
Prediction on the target city

Figure 9 Diagram representing the training script and domain adaptation process.

The training starts with pre-training the autoencoder and the domain discriminators using the source and target cities. As the decoder reconstructs the input, we can calculate a loss value composed of both a reconstruction criterion and an adversarial loss. After a determined number of epochs, the optimizer parameters are resettled, and the autoencoder is again trained with only the target city's data.

We freeze the autoencoder's weights for the second part and import the encoder block as the feature extractor. Again, the first step is to pre-train the predictor using the source cities' data and fine-tune it with the target city. For a more formal and structured explanation, Algorithm 1 exposes the training process of the autoencoder with domain discriminator, while Algorithm 2 does the same for the predictor block.

Since we use batch normalization as a regularization technique for all parts of our model, we selected ADAMW [43], a variation of ADAM, as the optimizer algorithm. This makes sense because ADAMW applies weight decay in a manner that is more compatible with batch normalization layers. Unlike the original ADAM optimizer, which adjusts gradients directly and can undesirably impact batch normalization's scale and shift parameters, ADAMW modifies the weights directly decoupled from the gradients, leading to more stable and consistent training dynamics.

Additionally, the utilization of GRADSCALER is noteworthy. This technique is employed to mitigate the underflow in gradient values during the backpropagation process, which occurs when the magnitudes of the gradients are so small that they are approximated to zero in the numerical precision being used. GRADSCALER addresses this issue by multiplying the loss value, from which gradients are derived, by a constant scaling factor. This operation increases the magnitudes of the gradients, preventing them from going to zero. After the backpropagation step, the gradients are then divided by the same scaling factor to restore their original magnitudes, preventing underflows without significantly impacting the performance.

To optimize the model's training time and required memory, as both are important resources, we make use of mixed-precision training, as proposed by [44]. With this technique, the operations during the training phase are done with half-precision, halving their size and enabling for the increase of the batch size. This goes well along with the GRADSCALER technique, as the less precise gradients may incur gradient vanishing.

Furthermore, it's interesting that we clarify the datasets used for each step of the training and evaluation of the model. During the pre-training phase, the autoencoder is trained on a combined dataset comprising both source and target data. This strategy facilitates the exposure of the domain discriminator to target features, which, in conjunction with the GRL, orchestrates the adversarial training of the autoencoder. The datasets from the source domains are uniformly sized, while the smaller target domain dataset is iteratively cycled to align with the larger source datasets. Also, to make sure that the Domain Discriminator is trained in an unbiased way, for each BATCH from the dataloader, we randomly select one city and use only this selected city and the target city to train the DD.

To enhance consistency, the pre-training of the predictor is also done on this same mixed dataset. The OPTIMIZER_PARAMS for the different tasks are different and tuned based on the behavior of the loss of each task over the epochs, which are also task-specific parameters to be tuned depending on the learning capacity of each part of the model.

During the fine-tuning phase, we reset the optimizers using new, smaller learning rate and weight decay values and focused on the target dataset only. For the autoencoder, we still train both the GRL and DD to ensure that the model can adapt to the specific characteristics of the target domain while preserving the domain-invariant nature of the features. Since the fine-tuning dataset is smaller and the learning rate is reduced, we expect this to refine the model's performance on the target domain while maintaining its ability to generalize across domains. For this training phase, the target city dataset comprised, at a base case, 2 weeks' worth of data (meaning 3360 data points) extracted from the whole dataset via random split.

All testing on the predictor is done with a never-seen target data dataset. This dataset consisted of all non-used data of the target city. As in our setup, we had the same amount of data for all cities (6 months or 43 200 data points). We used then ca. 40 000 data points for the testing.

Algorithm 1 Autoencoder with Domain Discriminator Training Process

```
Require: $AE\_params$, $DD\_params$, $num\_epochs$, $dataloaders$, $criterion$,
    $optimizer\_params$, $BATCH\_SIZE$, $lambda$
Ensure: Autoencoder $ae$, Domain Discriminator $dd$, Gradient Scaler $scaler$
 1: ENCODER, DECODER $\leftarrow$ AUTOENCODER($AE\_params$)
 2: $dd \leftarrow$ DOMAINDISCRIMINATOR($DD\_params$)
 3: $scaler \leftarrow$ GRADSCALER
 4: $optimizer \leftarrow$ ADAMW($\{ae, dd\}$, $optimizer\_params$)
 5: for $epoch = 1$ to $num\_epochs$ do
 6:     for each $dataloader$ in $dataloaders$ do        ▷ pre-training and fine-tuning dataloaders
 7:         for each $batch$ in $dataloader$ do
 8:             $total\_loss \leftarrow 0$
 9:             for each $data$ in $batch$ do        ▷ each batch can comprise $N$ different cities
10:                 $x, edge\_index, batch \leftarrow data$
11:                 $H \leftarrow$ ENCODER($x, edge\_index$)
12:                 $H \leftarrow$ GRADIENTREVERSALLAYER($H$)
13:                 $dan\_loss \leftarrow$ DOMAINDISCRIMINATOR($H, batch$)
14:                 $x\_recons \leftarrow$ DECODER($H, edge\_index$)
15:                 $feat\_loss \leftarrow criterion(x\_recons, x)$
16:                 $loss \leftarrow feat\_loss + lambda \cdot dan\_loss$
17:                 $total\_loss \leftarrow total\_loss + loss$
18:             end for
19:             BACKWARD($scaler.scale(total\_loss)$)
20:             STEP($optimizer$)
21:             UPDATE($scaler$)
22:             $optimizer$.zero_grad()
23:         end for
24:     end for
25: end for
```

Algorithm 2 Predictor Training Process

```
Require: $PR\_params, num\_epochs, dataloaders, criterion, optimizer\_params$
Ensure: Predictor $pred$, Autoencoder $ae$
 1: $pred \leftarrow$ PREDICTOR($PR\_params$)
 2: ENCODER $\leftarrow$ AUTOENCODER(*pre-trained*)        ▷ pre-trained and fine-tuned autoencoder
 3: $optimizer \leftarrow$ ADAMW($\{pred\}$, $optimizer\_params$)
 4: $scaler \leftarrow$ GRADSCALER
 5: for $epoch = 1$ to $num\_epochs$ do
 6:     for each $batch$ in $dataloaders$ do
 7:         $total\_loss \leftarrow 0$
 8:         for each $data$ in $batch$ do
 9:             $x, edge\_index, y, T\_features \leftarrow data$
10:             $H \leftarrow$ ENCODER($x$, $edge\_index$)
11:             $H \leftarrow$ FUSIONFEATURES($H$, $T\_features$)
12:             $y\_hat \leftarrow$ PREDICTOR($H$, $edge\_index$, $batch$)
13:             $loss \leftarrow$ CRITERION($y$, $y\_hat$)
14:             $total\_loss \leftarrow total\_loss + loss$
15:         end for
16:         BACKWARD($scaler.scale(total\_loss)$)
17:         STEP($optimizer$)
18:         UPDATE($scaler$)
19:         $optimizer$.zero_grad()
20:     end for
21: end for
22: function FUSIONFEATURES($H$, $T$)
23:     $T \leftarrow$ EXTENDS($H$.shape)
24:     $H \leftarrow H$ concatenated with $T$ along the feature dimension
25:     return $H$
26: end function
```

3.7. Loss Functions

In this Section, we define and explain the loss functions, also known as cost functions or criteria, that will be implemented or tested on the model or its parts. From the following criteria, MAE, RMSE and MAPE (Mean Absolute Percentage Error) are to be used exclusively as performance metrics. The purpose of listing and implementing a varied array of loss functions is to determine the best fit for our problem and data.

3.7.1. Mean Squared Error (MSE)

As the most known and traditional loss function, MSE is widely used in almost all fields of machine learning. In particular, it's one of traffic forecasting models' most "standard" loss functions as it focuses on minimizing significant errors.

$$\mathcal{L}(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{3.10}$$

3.7.2. Root Mean Squared Error (RMSE)

A typical variation of the MSE, RMSE is widely used in almost all fields of machine learning for testing as a performance metric. It is based on the same concept that governs its base function but on the same scale as the input data.

$$\mathcal{L}(y, \hat{y}) = \frac{1}{N}\sqrt{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{3.11}$$

3.7.3. Mean Absolute Error (MAE)

The MAE is not supposed to be used as a criterion during the training phase, but rather, it's widely used as a performance metric for testing. It calculates the average absolute difference between the predicted and actual values. It treats all errors equally and is particularly robust when dealing with outliers.

$$\mathcal{L}(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{3.12}$$

As it measures the average magnitude of errors in a set of predictions without considering their direction, the MAE offers a scale-dependent interpretation of how far off the predictions are, on average, from the actual values.

3.7.4. Mean Absolute Percentage Error (MAPE)

Similarly, MAPE is another performance metric to be applied during the testing phase. It uses the same logic as the MAE but calculates the percentage error of the absolute difference between the predicted and actual values. It's used in the same context as MAE, but it's easier to interpret and practical when dealing with data represented in different scales.

$$\mathcal{L}(y, \hat{y}) = \frac{100\%}{N} \sum_{i=1}^{N} \left| \frac{y_i - \hat{y}_i}{y_i} \right| \tag{3.13}$$

3.7.5. Weighted Mean Squared Error (WMSE)

As a variant of the MSE, the WMSE (Weighted Mean Squared Error) amplifies the significance of errors in certain parts of the dataset by multiplying each error by a specific weight. It's helpful in scenarios where certain data points are more critical than others and should have more influence on the total loss. Mainly, it's applied to heavenly imbalanced datasets, as is the case for the data used in this work. This criterion is also known as Zero Inflation Loss when defining a weight for zero values.

$$\mathcal{L}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} w \cdot (y_i - \hat{y}_i)^2 \tag{3.14}$$

3.7.6. Mean Squared Logarithm Error (MSLE)

This other variation of the MSE criterion consists of the mean of the squares of the logarithmic differences between the predicted and actual values. This criterion reduces the impact of significant errors on large true values. They are applied to problems where the target values have a wide range.

$$\mathcal{L}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (\log(y_i + 1) - \log(\hat{y}_i + 1))^2 \tag{3.15}$$

3.7.7. Weighted Mean Squared Logarithm Error (WMSLE)

The WMSLE (Weighted Mean Squared Logarithm Error) is a sophisticated criterion that combines the aspects of the MSLE (Mean Squared Logarithm Error) and the WMSE. It addresses the challenge of imbalanced datasets, focusing on penalizing the errors associated with non-zero targets more heavily.

This loss function is particularly adept at handling datasets where the prediction of non-zero values is more crucial than the prediction of zeros, which may be abundant but less informative. The WMSLE is therefore especially useful when the cost of an error varies depending on the magnitude of the true value, such as in datasets with many zero entries but where accurate prediction of the non-zero values is paramount, which happens to be the case in our dataset (as discussed in Section 3.1). The WMSLE is defined as follows:

$$\mathcal{L}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} w \cdot (\log(y_i + 1) - \log(\hat{y}_i + 1))^2 \tag{3.16}$$

3.7.8. Custom Huber Loss

The Custom Huber Loss combines the MSE and MAE. It's quadratic for small errors and linear for significant errors. We can also apply a Zero Inflation Loss for zero target values. It's advantageous when dealing with outliers and versatile as it can have the best of both criteria on the domains they are the best.

$$\mathcal{L}_{\delta,w}(y, \hat{y}) = \begin{cases} \frac{1}{2} w \cdot (y_i - \hat{y}_i)^2 & \text{for } y = 0, \\ \frac{1}{2}(y - \hat{y})^2 & \text{for } |y - \hat{y}| \le \delta, \\ \delta \cdot (|y - \hat{y}| - \frac{1}{2}\delta) & \text{otherwise.} \end{cases} \tag{3.17}$$

3.7.9. Log-Cosh Loss

The Log-Cosh error is defined by the logarithm of the hyperbolic cosine of the difference between the prediction and actual values. It has a shape similar to the MSE but is smoother and less sensitive to outliers. It's useful as it has the robustness of the MAE while maintaining the smooth gradient of the MSE. This loss function is differentiable, unlike MAE.

$$\mathcal{L}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} \log(\cosh(\hat{y}_i - y_i)) \tag{3.18}$$

3.7.10. Binary Cross Entropy

In the binary classification, the BCE loss penalizes deviations from the actual labels by comparing the predicted probabilities against the ground truth, providing a robust gradient signal for model updates during training.

$$\mathcal{L}(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log(\sigma(\hat{y}_i)) + (1 - y_i) \log(1 - \sigma(\hat{y}_i))] \tag{3.19}$$

3.8. Baselines

To better understand the performance achieved by the models elaborated, we propose using three different baselines: HA (Historical Average) and ARIMA (Autoregressive Integrated Moving Average), as statistical models, and ConvLSTM, as a deep learning one. In this Section, we explain each of the baselines, how they are calculated, and how they are applied to the dataset.

It's noteworthy that the dataset used to calculate the baselines is the target dataset only, as the main problem to be tackled on this thesis is how to predict future states of the target domain.

To avoid the effects of biases related to the unbalance of the data, the values taken to calculate the metrics on the historical average and ARIMA are the non zero ones. Since the data collection is done with probes, there are huge parts of the city that are not visited regularly by the data collectors and thus yield 0 for every time step.

3.8.1. Historical Average

As one of the most common baselines in this field, HA is based on the premise of temporal consistency, leveraging the average of data from corresponding time periods in the past—such as previous weeks—to forecast current traffic conditions.

The HA would receive, as input, N weeks worth of data to "train". This means that when comparing to a transfer learning model that has a training dataset comprising 2 weeks of target data, the average would be composed of the last 2 values (i.e. $N = 2$). For a 1 week target dataset, this baseline just repeats the values from last week.

3.8.2. ARIMA

As the other most famous and employed non-transfer baseline model, ARIMA is a statistical time-series regression based on autoregressive (AR) and moving average (MA) components, which account for past values and errors, respectively, along with an integrated (I) element for differencing to induce stationarity [45].

Similarly to the historical average, the ARIMA baseline is going to be "trained" (or rather fitted) using the same target data that will be used to train the transfer learning model.

3.8.3. Convolutional LSTM (ConvLSTM)

Similar to the GConvLSTM we used to develop our autoencoder, the ConvLSTM architecture is an already established composition that aims to extract both spatial and temporal features. The difference between these two modules is that the later lacks the graph component, and therefore can't make use of the adjacency matrix or connectivity information when extracting features.

The implementation we used for this purpose is the one made by [46], which was based on the works of [47]. The model developed for this baseline is composed of three ConvLSTM layers. Their main purpose is to expand the latent space of the input, from `num_channels` to a certain `latent_dim`, hyperparameter of the model. In our use case, we had cells with latent dimensions of $8, 16, 32$ and a kernel size of 3. The last hidden state of the last layer is then passed through a ReLU (Rectified Linear Unit) activation, and a linear layer brings the output of the activation function back to the number of desired channels, and a Sigmoid is used after all steps as a normalization layer.

4. Results

4.1. Computational Specifications and Training Times

This section specifies the computational hardware used during the development of the model. Here, we detail the machines employed during training and testing and the time each part of this process consumes. Table 3 shows the GPU models used and the training times for each dataset configuration and part of the model. The RTX 3070 was available on a personal computer, while the Tesla T4 and RTX 4090 were run as servers. The availability of the computational resources in the workspace determined the usage of different machines. Besides the time spent on training, the model's performance does not depend on the GPU. The headers “Central Square” and “Whole City” should be interpreted as the “Time to train - (in minutes, per epoch per source city) in this dataset configuration”.

GPU Model	**VRAM (GB)**	**Central Square**		**Whole City**	
		AE	**PRED**	**AE**	**PRED**
RTX 3070	8	150	75	-	-
Tesla T4	16	60	30	660	480
RTX 4090	24	40	20	330	240

Table 3 Hardware specifications and training times. These training times are computed as an approximation of the average time required to train the model in each configuration.

As observed in Table 3, the training process is very time-consuming, even for powerful graphic cards. This is why, for the preliminary experiments, we chose to use only the Central Square of each city.

4.2. Autoencoder Experiments

This Section will present the experiments on the variation of the autoencoder's parameters. Experiments 1, 2, 3, 4, and 5 are related to the hyperparameter search for the autoencoder, as we tested for the impact of the K_{cheb}, the number of source cities, the activation function, the criterion function, and the latent dimensions of the autoencoder. Experiment 6 explores parameter sharing as a domain adaptation technique for the autoencoder.

Table 4 shows the parameters used for most experiments in this Section. Note that when a different value is used for a determined experiment, this value would be stated on a similar table in the experiment explanation.

Parameter	**Value**
Batch size	32
Number of cities	2
Epochs	2
Chebyshev polynomial degree	2
Convolution dimension	16
Linear dimension	8
Activation function	ReLU
Dropout rate	0.5
Loss criterion	ZeroInflationLoss($w = 100$)

Table 4 Fixed parameters for the autoencoder experiments

Experiment 1: Impact of the Chebyshev polynomial degree parameter on the autoencoder's performance

The Chebyshev polynomial degree is one of the parameters of the `GConvLSTM` cell used as the backbone of the autoencoder. It dictates the receptive field of the model, as it limits the number of neighbors that will be used to structure a computational graph during backpropagation. For instance, for a value of $K_{\text{cheb}} = 1$, the computational graph will have a depth of 1, and only the nodes individually will be part of it. A $K_{\text{cheb}} = 2$ implies that the computational graphs will contain not only the nodes themselves but the immediate neighbors of each node.

In this experiment, we evaluated the autoencoder's performance with varying values of K_{cheb}, ranging from 1 to 6, while maintaining all other parameters as specified in Table 4. Figure 10 illustrates the distributions of both MAE and MSE for the corresponding K_{cheb} values. The boxplots reveal a discernible trend where both MAE and MSE metrics decrease as K_{cheb} increases from 1 up to 3, indicative of improved performance. The median of MAE reaches its minimum at $K_{\text{cheb}} = 3$, suggesting this is the optimal Chebyshev polynomial degree for capturing the ST features within the dataset, given the current parameter configuration. However, for K_{cheb} values greater than 3, there is an observable increase in variance and a slight rise in error metrics, which may signal a risk of overfitting.

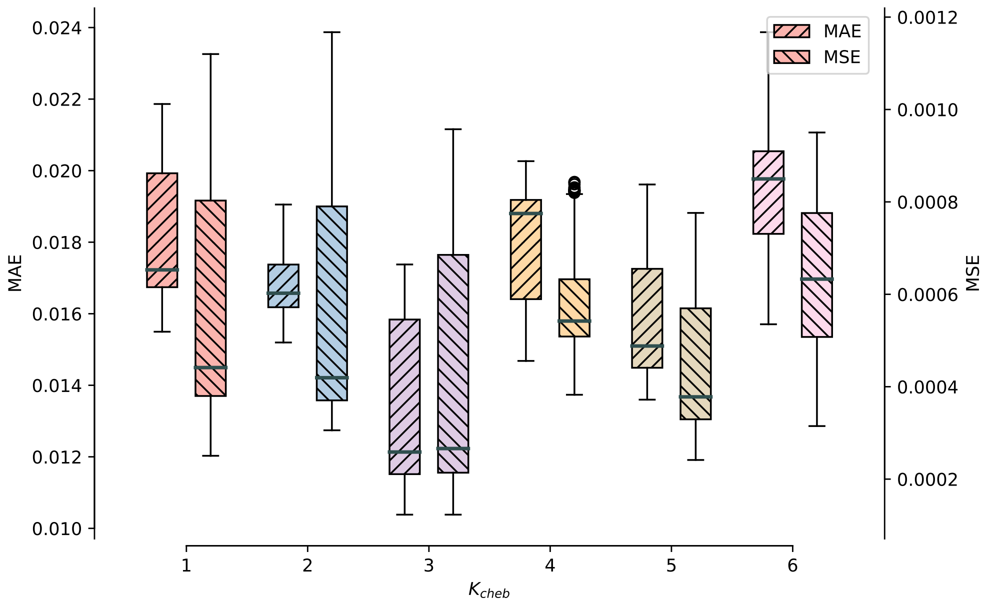

Figure 10 MAE and MSE for the autoencoder with different values of K_{cheb}

Parameter	**Value**
Batch size	8

Table 5 Specific parameters for Experiment 2. We use “Batch size=8” due to GPU memory constraints.

Experiment 2: Impact of the number of source cities on the autoencoder’s performance

Experiment 2 deals with the impact of the number of cities on the autoencoder’s performance. The number of cities indicates the data available to the model and the diversity and complexity presented. Four different values for the number of cities were considered: 1, 2, 4, and 8. The fixed parameters of the experiment are presented in Tables 4 and 5, and Figure 11 shows the results obtained.

For training over just one city $(N_{cities} = 1)$, we observe a relatively high median value for both MAE and MSE, with a wide IQR (Interquartile Range), suggesting that the model lacks generalization capabilities compared to the other configurations. Introducing another city to the model $(N_{cities} = 2)$ seems to result in a slighter better median for the MSE, but at the cost of outliers and a significantly higher median value for the MAE.

When training over four $(N_{cities} = 4)$ and eight $(N_{cities} = 8)$ cities, the model seem to behave similarly. It’s noticeably better than the previous models when comparing the median values for both metrics, indicating better performance and generalization capabilities. There are outliers at $(N_{cities} = 8)$, which may indicate a limit on how much diversity can be inputted into the model while trying to increase generalization capabilities.

The overall trend suggests that incorporating data from more cities enables the model to learn more generalizable features across different domains, reducing the model's bias toward specific cities' traffic patterns. It's worth noting that increasing the number of cities also increases the computational cost of training the model. Therefore, using the N_{cities} values near 4 seems more optimal.

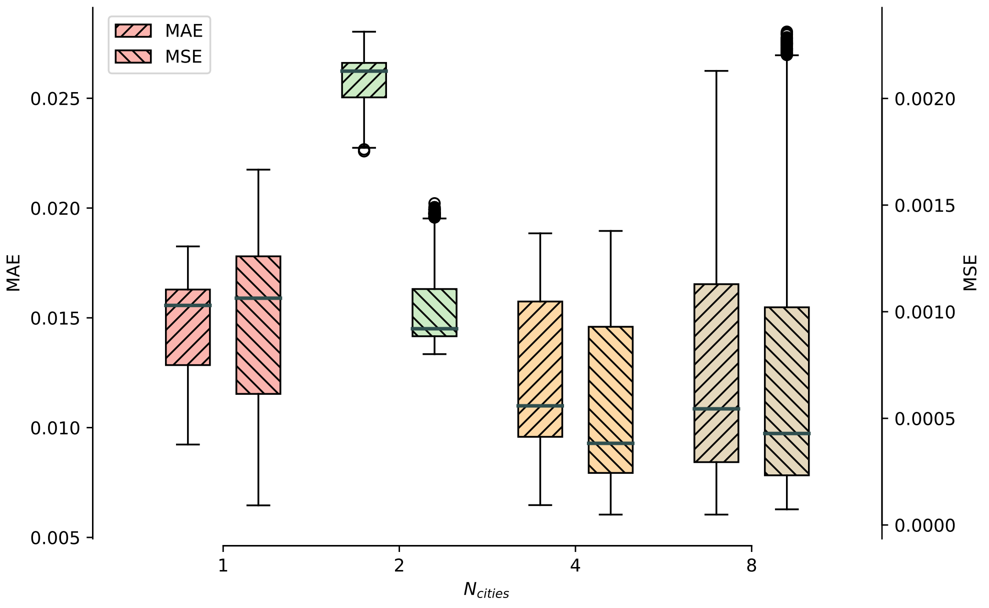

Figure 11 MAE and MSE for the autoencoder with different number of cities

Experiment 3: Impact of the activation function on the autoencoder's performance

Parameter	Value
Chebyshev polynomial degree	3

Table 6 Specific parameters for Experiment 3. We use $K_{\text{cheb}} = 3$ as this is the optimal value found in Experiment 2.

The choice of activation function within neural network layers substantially impacts the model's ability to capture and represent complex patterns within the data. This experiment examines how the activation function selection influences the autoencoder's performance. Three standard activation functions were considered: ReLU, Sigmoid, and TanH (Hyperbolic Tangent).

For this analysis, we trained separate autoencoder models using each activation function while keeping all other parameters constant, as specified in Tables 4 and 6. Figure 12 presents the distributions of MAE and MSE across the different activation functions.

The boxplots indicate that the model with the ReLU activation function exhibits a slightly higher median MAE than the other functions but also has a broader IQR, suggesting less consistent predictions across different instances. On the other hand, the model utilizing the Sigmoid function shows a tighter IQR in MAE, indicating less variability in its predictions. However, its median MAE is higher. Interestingly, the TanH function yields the lowest median

MAE, suggesting it may be the most effective at capturing the underlying ST patterns for this dataset and model configuration.

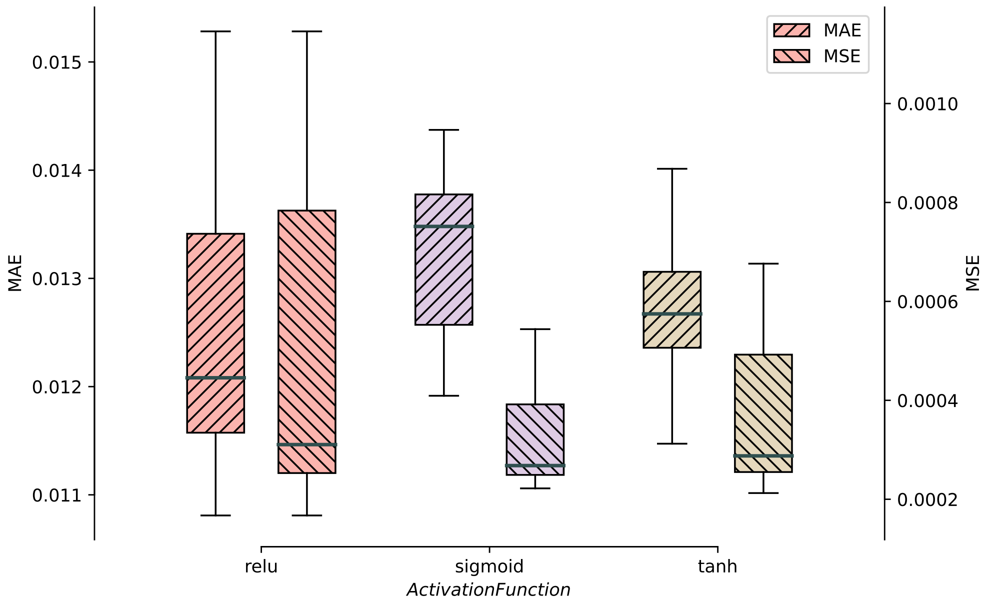

Figure 12 MAE and MSE for the autoencoder with different activation functions.

Experiment 4: Impact of the criterion function on the autoencoder's performance

Parameter	Value
Chebyshev polynomial degree	3

Table 7 Specific parameters for Experiment 4. We use $K_{\text{cheb}} = 3$ as this is the optimal value found in Experiment 2.

Another interesting choice that can be made in the autoencoder training is the criterion function that yields the backpropagated loss. In this experiment, we analyze how different loss functions can influence the autoencoder's performance. All criteria used here were presented and explained in Section 3.7. All other parameters are exposed in Tables 4 and 7. The losses considered were:

- MSE
- WMSE $(w = 10)$
- WMSE $(w = 100)$
- MSLE
- WMSLE $(w = 10)$
- WMSLE $(w = 100)$
- Log-Cosh Loss

- Focal Loss $(\alpha = 0.25, \gamma = 2)$

Figure 13 provides a comparative overview of the performance of these criteria on the autoencoder. From the plots, it is evident that both Focal Loss (with the input parameters) and Log-Cosh Loss are worse than their pairs, showing more considerable overall error (MAE and MSE) and IQR, indicating that they are not fit for being used as criteria.

The traditional MSE provides a baseline for comparison, exhibiting a moderate spread in MAE values. The MSLE is designed to be less sensitive to significant errors by emphasizing the logarithmic difference between the predicted and actual values, which is evident in the lower median MAE it achieves compared to MSE.

A more nuanced approach is observed with both WMSE and WMSLE, where introducing a weight factor (w) aims to penalize errors differently based on their magnitude. Notably, for the WMSE, as the weight increases from $w = 1$ (MSE) to $w = 10$ to $w = 100$, the spread and median of the MAE decrease, suggesting a tighter grouping of errors around a lower central value. However, an increase in weight also introduces a higher variance in MSE, as indicated by the presence of outliers, particularly for $w = 100$. This implies that while WMSE can potentially reduce the average error, it may also lead to more extreme errors in some instances. For the WMSLE criterion, a similar result is observed, with the increase of w being associated with a smaller IQR but a higher median of the MAE.

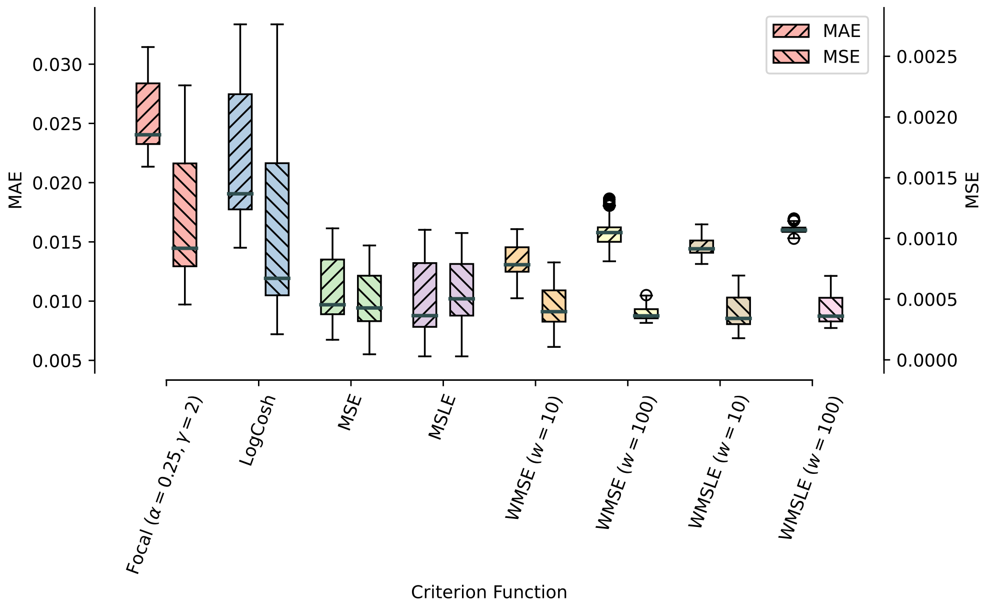

Figure 13 MAE and MSE for the autoencoder with different criterion functions.

Experiment 5: Impact of the latent dimension's size on the autoencoder's performance

Experiment 5 focuses on the impact of inner (or latent) layer dimensions on the autoencoder's performance. For this purpose, several pairs of convolutional and linear dimensions were

Parameter	**Value**
Chebyshev polynomial degree	3
Activation function	TanH

Table 8 Specific parameters for Experiment 5. We choose them parameters based on the optimal parameters found in previous Experiments.

tested and are exposed in Table 9. Furthermore, Tables 4 and 8 show the fixed parameters for the experiment.

Convolutional Dimension	**Linear Dimension**
8	2
8	4
8	8
16	4
16	8
16	16
32	8
32	16

Table 9 Tested pairs of convolutional and linear dimensions.

Analyzing Figure 14, it's possible to note that the convolutional dimension is the most critical parameter between the two. Models with smaller convolutional dimensions ($\texttt{conv\_dim} = 8$) tend to have higher MAE and MSE values, indicating a lower performance. This suggests that these models may not have enough complexity to capture the data's relevant ST features.

Going up to $\texttt{conv\_dim} = 16$, we can observe a significant reduction in both metrics. This points to a better extraction feature, enabling the autoencoder to capture more of the data's complexity. Finally, with $\texttt{conv\_dim} = 32$, we note an increase of the median and extremes of the error, but a smaller IQR, suggesting that we've reached a point where overfitting starts to take effect on the model.

Considering the values of the linear dimensions now, there's no clear trend that we can extract from the results, as an increase in it for low values of convolutional dimension led to more significant errors, but for other values, it made no difference. It's important to note that increasing convolutional and linear dimensions implies higher model complexity and, thus, higher computational cost for training it. For this reason, settling for pairs like $(16, 4)$ or $(16, 8)$ is the best option for balancing performance with cost.

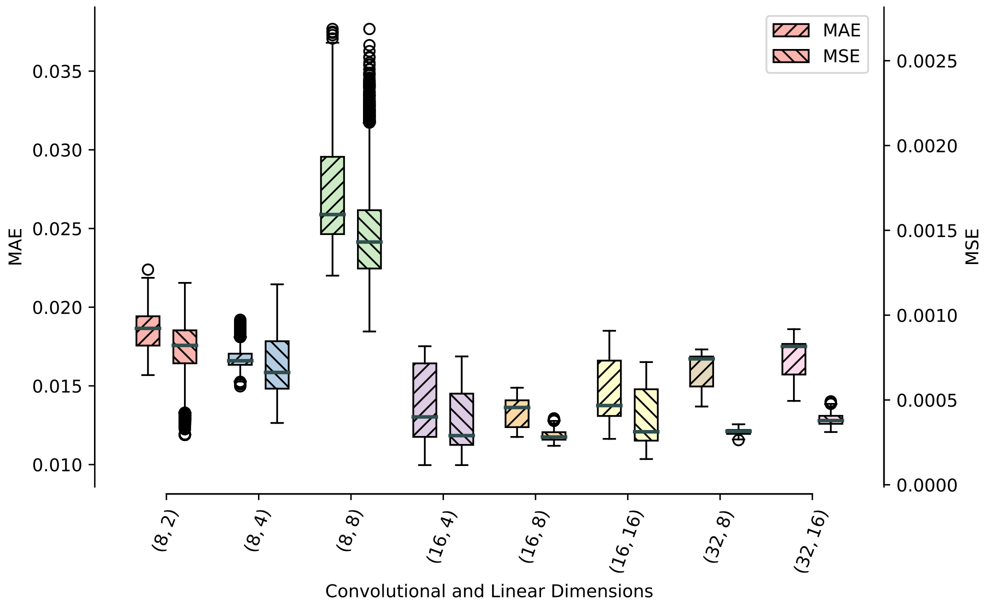

Figure 14 MAE and MSE for the autoencoder with different pairs of latent dimension values.

Experiment 6: Pre-training as a domain adaptation technique for the autoencoder

Parameter	Value
Batch size	16
Number of cities	1
Epochs	4
Chebyshev polynomial degree	3
Activation function	TanH

Table 10 Specific parameters for Experiment 6. We use a reduced batch size and only one city due to memory constraints. Other parameters are derived from previous Experiments.

Experiment 6 is the first attempt to apply a fine-tuning technique to transfer knowledge from one model to another. It consisted of training the model across three different setups: using solely the target city data (“Target Only”), pre-training with data from one source city followed by fine-tuning on the target city data (“Pre-trained w/ 1 Source”), and pre-training with data from two source cities before fine-tuning on the target city data (“Pre-trained w/ 2 Sources”). The results, illustrated in Figure 15, underscore the benefits of transfer learning through pre-training.

The “Target Only” model shows the higher median and IQR for both MAE and MSE when compared to the pre-trained ones. Conversely, both pre-trained setups significantly reduce both error metrics' median and IQR. This suggests that the pre-training is an effective method for domain adaptation, as the model gets significantly better at generalizing and accurately predicting the ST patterns. Furthermore, the image also indicates that having more cities can enhance this technique, as the diversity of data presented in the model increases its generalization capacity, making it more accurate.

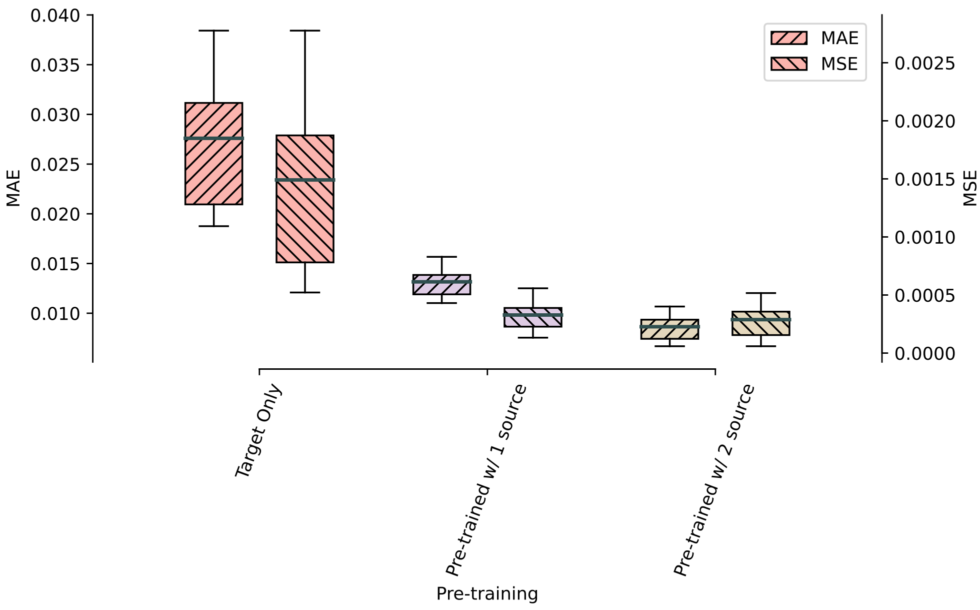

Figure 15 MAE and MSE for the autoencoders using different pre-training strategies.

4.3. Predictor Experiments

Given then an autoencoder capable of effectively extracting the relevant ST features, which we found with the previous experiments, it's necessary now to define good hyperparameters for the predictor part of the model. Experiments 8, 9, and 10 investigate the predictor's optimization by varying the relevant parameters of the model and datasets, such as the latent dimensions, the number of training epochs, and the quantity of target data available while Experiment 7 studies the efficacy of the adversarial training.

Similarly to how we organized the previous Section, Table 11 shows the fixed parameters for all predictor experiments. If a parameter from this table is to be changed, it will be clearly stated on the particular experiment's specific parameters table.

Experiment 7: Impact of the lambda regularization on the predictor

As we complete the development of the autoencoder, it's essential to address one of the main challenges of the proposed model: to generalize the knowledge obtained on the source domain to the new, different target domain. As presented in the Methodology section, we aim to learn how to extract domain-invariant features, which hold the key to a model's ability to

Module	Parameter	Value
both	Batch size	64
both	Epochs	2
both	Number of source cities	1
both	Weeks of target data	2
AE	Chebyshev polynomial degree	3
AE	Convolution dimension	16
AE	Autoencoder Linear dimension	8
AE	Activation function	TanH
AE	Dropout rate	0.5
AE	Loss criterion	MSE
AE	λ regularization	0.1
PRED	Activation function	ReLU
PRED	Loss criterion	MSE
PRED	Top-K Pooling	1750
PRED	Linear Dimension	32

Table 11 Fixed parameters for Predictor experiments

perform well across varied domains without succumbing to domain-specific biases.

In this context, Experiment 7 was designed to investigate the impact of balancing the reconstruction loss of the autoencoder with the domain discriminator loss, modulated by a regularization parameter λ. This parameter regulates the trade-off between the fidelity of the reconstruction and the degree of domain invariance imposed by the DD. Since we're introducing a regularization technique in the model's training, we expect that the individual performance of the autoencoder must pay for this enhanced performance on the generalization of the features, as we try to “fool” it during training.

The values of λ tested range from 0 to 1, including $0.001, 0.005, 0.01, 0.1, 0.5,$ and 1. A value of 0 means no regularization from the domain discriminator and, therefore, no adversarial training, while a value of 1 means that reconstruction and domain discriminator losses have the same weight when composing the total loss.

Figure 16 exposes the results obtained from training these models. The plot suggests that including the domain discriminator loss positively impacts the model's prediction ability as the performance for $\lambda = 0$ is inferior to that for $\lambda \neq 0$. In particular, we can observe that for the MAE boxes, there's an apparent decrease in both median and max values when going from $\lambda = 0$ to $\lambda = 0.001$ and then from $\lambda = 0.001$ to $\lambda = 0.005$, from which point it's hard to observe significant changes on the performances. However, one can verify that the number of outliers on the maximum size of the boxes seems to increase.

One possible explanation for the lack of changes observed from a certain point onwards is that as we reach a point of stabilization during training, the reconstruction loss starts to reach low values (in the order of 10^{-4}). At the same time, we maintain the domain discriminator loss oscillating around 0.6 (the ideal expected value for it is $\ln(2) \simeq 0.693$, which occurs when the domain discriminator can't identify the domain due to the features being completely domain-agnostic). This means that the domain discriminator loss, supposed constant, will dominate the total loss almost constantly, and, for a determined threshold, there's no difference between $\lambda = 0.5$ and $\lambda = 1$ as they both result in a seemingly significant loss for the optimizer.

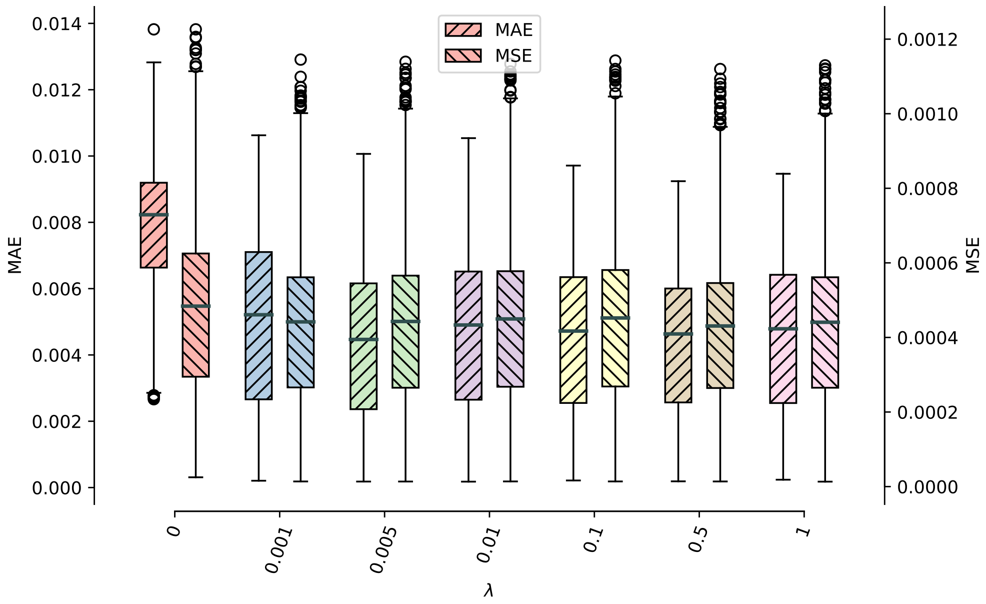

Figure 16 MAE and MSE on the prediction using different values of λ.

Experiment 8: Impact of the linear layer dimension on the predictor

Experiment 8 was projected to better understand the impact of the linear layer dimension, the output size of the A3T-GCN layer, on the model's performance. The experiment assessed four sizes for the linear dimension: 8, 16, 32, and 64. Figure 17 shows the MAE and MSE distribution for the different compositions.

The trend observed in the experiment is clear: with the increase of the latent dimension, the medians for MAE decrease significantly, while the ones for MSE decrease slightly. Noteworthy is the decrease on the maximum of the MSE boxplots. Overall, the results indicate that the increase on the size of the latent dimension imply a increase on the generalization capabilities of the model.

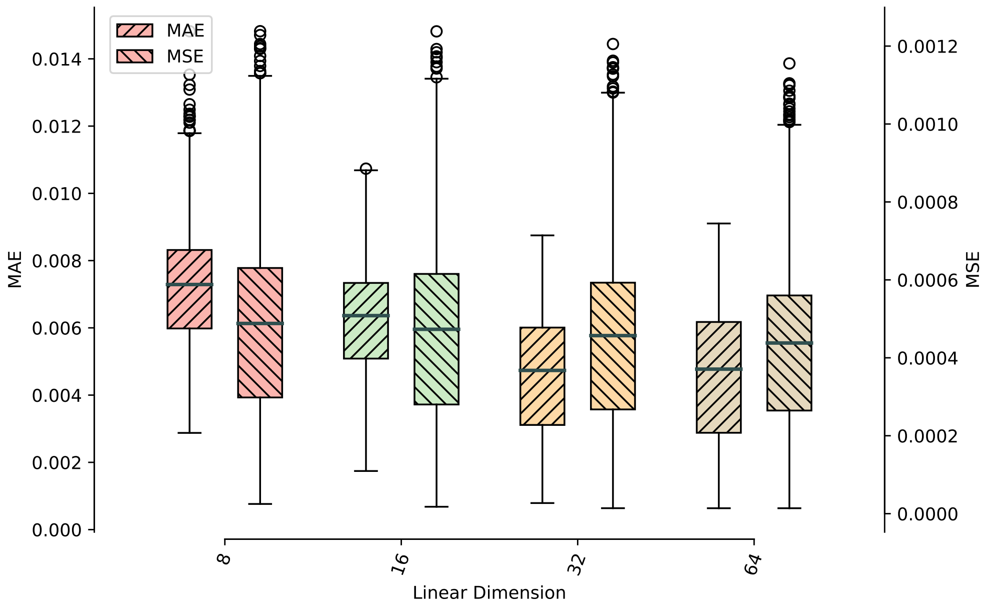

Figure 17 MAE and MSE on the prediction using different domain adaptation techniques.

Experiment 9: Impact of the number of epochs on the predictor

In machine learning terms, an epoch represents a complete cycle through the data, providing the model with repeated exposure to the training examples and allowing it to incrementally adjust its parameters. Changing the number of epochs to be used impacts the model's performance directly, as an epoch count that is too low may lead to underfitting. At the same time, too many epochs can cause overfitting. The investigation proposed in this experiment is critical for understanding the temporal dynamics of model convergence and identifying the most resource-efficient training regimen that maintains or enhances model efficacy.

For this experiment, we tested the model proposed using five different values of the number of epochs, ranging from 1 to 5, to be used on the autoencoder and predictor training. Figure 18 contains the results obtained. There's a clear trend of improvement in the model's performance as we increase the number of training epochs. We can observe a considerable gap on both MAE and MSE from one to two epochs. Still, the rate of improvement appears to diminish after that, as the median for the remaining boxes seems to share similar medians and IQR. Nonetheless, we can observe that the maximum error (for both metrics) continues to decrease with the increase of epochs. This indicates that the model is becoming more robust to outliers and noise in the data.

As we reach three training epochs, we've got a plateau regarding median error for the metrics, suggesting that the model may be approaching its learning capacity given the provided data. As we're dealing with a relatively large training dataset (in both numbers of data points and information/nodes per data point), it's plausible to consider that with only three epochs, the model has effectively learned how to capture the underlying patterns in the data. Since, as discussed in Section 4.1, an epoch of training with one source city takes approximately two hours, given the computational resources available, we can consider that the modest number of three epochs is sufficient for the model to learn. We can allocate the processing power into other areas of the model, such as supporting more source cities.

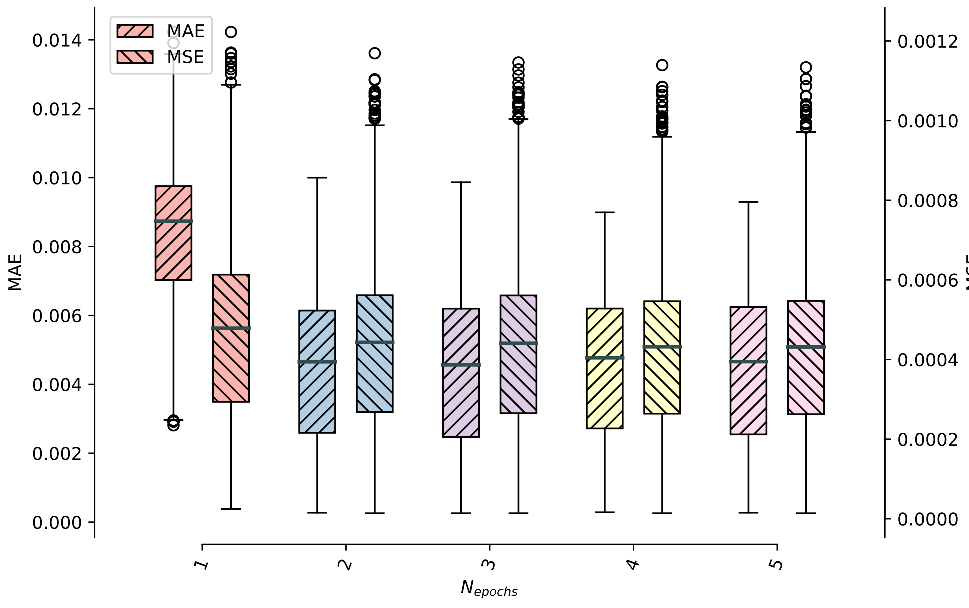

Figure 18 MAE and MSE on the prediction using different number of epochs.

Experiment 10: Impact of the scarcity of the target dataset

As the volume of available target data is one of the main grounds on which the development of transfer learning models is based, we find it interesting to experiment with how the availability of this kind of data impacts the overall performance of the model. By incrementally increasing the quantity of target data from 1 to 4 weeks, we aim to uncover how the breadth of target domain exposure affects the model's capacity to learn and generalize. Also, for many decision-makers, for which such models must yield important information that will govern their decisions, the availability of such data has a monetary price. By examining the model's performance with 1, 2, 3, and 4 weeks of target data, we can evaluate the prediction performance one can obtain for that price.

From Figure 19, we observe that as the quantity of target data increases from 1 to 4 weeks, there is a general trend of improvement in prediction accuracy. This is evidenced by the overall decrease in MAE, suggesting that the additional data helps the model to make more accurate forecasts. The MAE values appear to show the most significant decrease between 1 and 2 weeks of data, after which the improvements become less pronounced.

The MSE values, which are more sensitive to outliers due to squaring the errors, remain relatively stable across the different weeks. The presence of outliers in the MSE indicates that there are still instances where the model's predictions deviate substantially from the actual values, regardless of how much data it has been trained on.

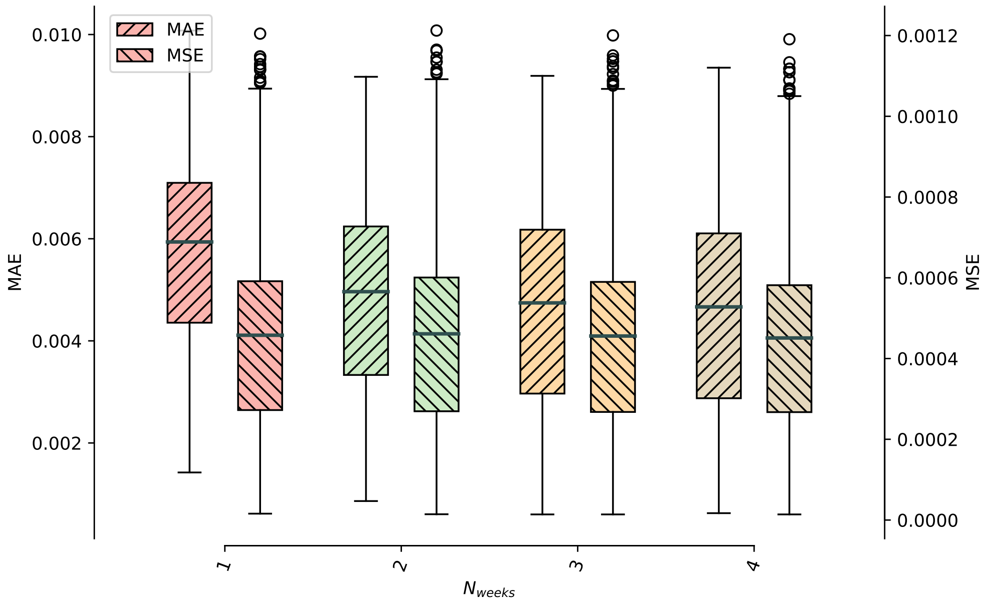

Figure 19 MAE and MSE on the prediction using different number of weeks for the target dataset.

4.4. Full Model Experiments

Building upon the insights obtained from the previous experiments, we now assess the entire model, experimenting with the complete model in its best possible form. Also, to test the model in its best extension, for this Section, we use the entire city's data, not only the central square we've been using until now to save computation resources (as discussed in Section 3.1).

The selected target city to be considered in all experiments done is Melbourne. The target cities for the transfer learning methods are Antwerp, Bangkok, and Barcelona. They were chosen at random from the available cities on the dataset.

Furthermore, for this section, the trained models will be tested against the following baselines: ARIMA and HA, standard statistical baselines in traffic prediction, ConvLSTM, and our model in data-driven mode, meaning that no sources are used for pre-training. All baselines and models are evaluated by two metrics: MAE, and RMSE, which are both frequently used as benchmarks for models on the field.

4.4.1. Model Setup

As we move to prepare the models, it's fundamental that we clearly define all the parameters used on what we call the "Full Model". We derive them from all previous experiments run for both the autoencoder and predictor modules. Table 12 presents these parameters, categorized by the specific part of the model they pertain to. As we are now using the full city data on training and prediction, the size of the tensors are at least 10 times bigger. Consequently, even with more powerful GPUs, the maximum manageable batch size is limited to 2.

With this constraint, we propose the use of Gradient Accumulation [48], meaning that we calculate the gradient at every iteration using OPTIMIZER.BACKWARD(), but only update the optimizer every N_{ga} steps. This makes a dataloader with batch size of 2 and $N_{ga} = 32$ produce the same effect on the optimizer during training as a dataloader with batch size of 64.

Additionally, as the batch size is reduced, the application of BATCHNORM stops to make sense, as we can't compute meaningful statistics with a sample size of 2, and no statistics at all for the extreme case where batch size is 1. We propose the substitution, for this experiments, of the BATCHNORM layer for the LAYERNORM [49]. This normalization operation normalizes an input tensor across the features' dimension.

4.4.2. Results

Table 13 contains the resulting median of the metrics for our model in six different configurations, with the variation of two hyperparameters: the number of sources used to pre-train the model (1, 2, or 3) and the number of weeks of available fine-tuning data (1 or 2). Furthermore, we also present in the same table the results of the baselines for two available data configurations (1 or 2 weeks).

The results presented show that the Transfer Learning approach has overall superiority in relation to the baselines, both the statistical and data-driven ones. Particularly, we can observe a clear enhance on performance in relation to a fine-tuning only approach using the same model, but limiting training data to only the target dataset.

Furthermore, Figure 20 shows the resulting distributions of the prediction errors for the six configurations tested. We note that there seems to be no substantial change when considering all six configurations. A small disturb can be observed, but the overall distributions seem to be very close, suggesting that the parameter variation has no impact on the model's performance.

The model also seems to be robust in relation to the number of training weeks available for fine-tuning. This indicates that the amount of data contained in one week (ca. 1700 datapoints) is already enough for the model to learn the specific target patterns.

Module	Parameter	Value
both	Batch size	2 or 1
both	Epochs	2
AE	Chebyshev polynomial degree	3
AE	Convolution dimension	16
AE	Autoencoder Linear dimension	8
AE	Activation function	Sigmoid
AE	Dropout rate	0.5
AE	Loss criterion	MSE
AE	λ regularization	0.01
AE	Optimizer Learning Rate	5×10^{-4}
AE	Optimizer Weight Decay	5×10^{-5}
PRED	Activation function	ReLU
PRED	Loss criterion	MSE
PRED	Top-K Pooling	1750
PRED	Linear Dimension	32
PRED	Optimizer Learning Rate	5×10^{-4}
PRED	Optimizer Weight Decay	5×10^{-5}

Table 12 Fixed parameters for the Full Model experiments

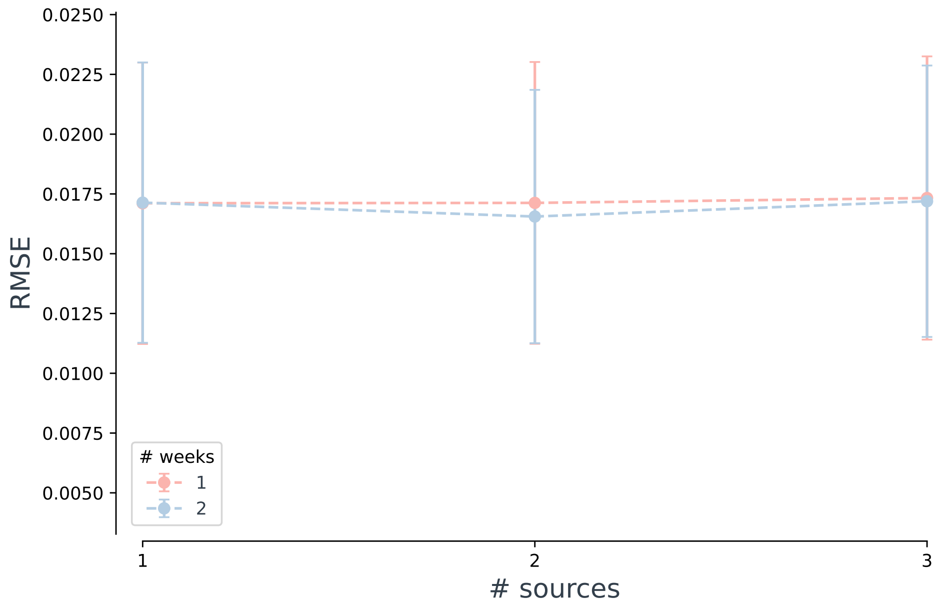

Figure 20 RMSE on prediction for the full model

		1 week						2 weeks					
		RMSE			MAE			RMSE			MAE		
Statistical	ARIMA	0.0821			0.0523			0.0785			0.0499		
Statistical	HA	0.1227			0.0947			0.1150			0.0888		
Data-Driven	ConvLSTM	0.1373			0.0853			0.1227			0.0847		
	ours (FT only)	0.1695			0.0404			0.1504			0.0356		
Transfer	# Sources	1	2	3	1	2	3	1	2	3	1	2	3
Learning	ours	0.0171	0.0171	0.0173	0.0018	0.0017	0.0019	0.0171	0.0166	0.0172	0.0017	0.0020	0.0036

Table 13 Results of the model and proposed baselines

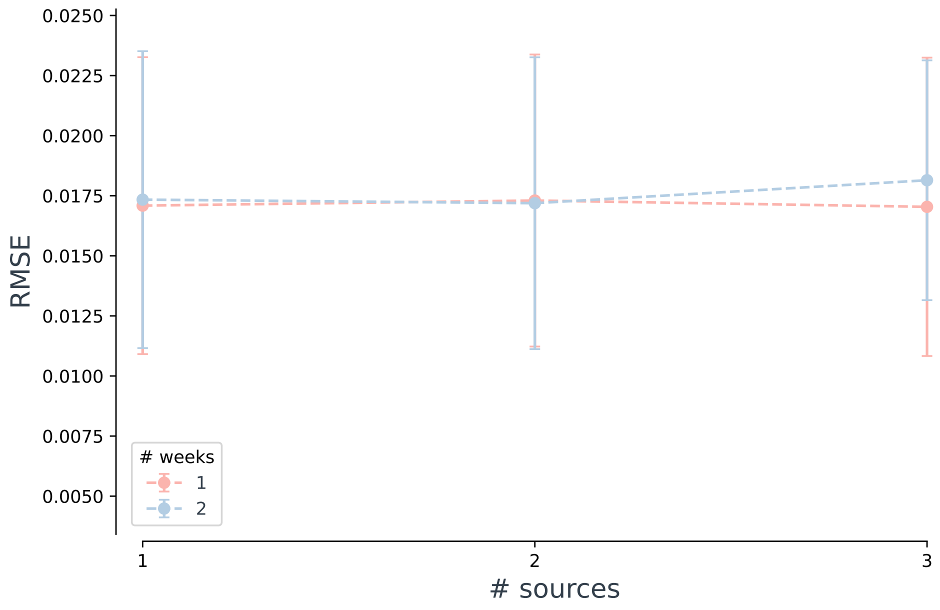

Figure 21 RMSE on reconstruction for the full model

Investigating further, the same pattern can be observed on the reconstruction RMSE for all six configurations, as shown in Figure 21. It seems that the increase on the number of sources doesn't increase the model's capacity to better learn domain-agnostic patterns on the dataset. Additionally, the use of one or two weeks of data seems to not imply any effects on the model's capacity, as the error values are very close, and the distribution bars seem to coincide in most of their ranges.

5. Discussion

5.1. On the hyperparameter search

For the hypeparameter search task, we tested several model configurations, parameters, and adaptation techniques. The ten most interesting experiments were exposed in Sections 4.2 and 4.3. As we built the model iteratively, the pursue of validation steps to prove or disprove propositions made or concluded during the development of the methodology.

Some of the main highlights we found on these experiments helped us guide the next step on the research and enhance our understanding on both the problem and the model's components. For instance, Experiment 7 helped us prove the positive effect of the domain discriminator on the adaptation of the model to the target domain. Experiment 4, on the other hand, showed us that selecting a good criterion function can help the learning of the model substantially.

Overall, with the best configuration assembled, and without the presence of the domain adversarial training (and GRL), we were able to reach RMSE values of under 2×10^{-2}, suggesting that the autoencoder is capable of reconstructing the provided input fairly well.

Furthermore, in Experiment 7 we studied the impact of the λ regularization hyperparameter and found that the Domain Discriminator in combination with a GRL produces better results than a model without it, supporting the thesis on its effectiveness.

Another interesting finding was obtained in Experiment 6, as we tested pre-training as a domain adaptation mechanism. The results obtained hint that pre-training is indeed a effective method for domain adaptation, and that an increase in the number of source cities can increase the model's performance significantly.

5.2. On the model's performance

The results obtained on the Full Model Experiment show that the obtained performance is well above the proposed baselines. Considering a 1-week fine-tuning dataset, we can observe enhancements on the range of 70-90% depending on the baseline considered compared to the model using only one source city.

These resulting metrics are well in accord to what we observed during the initial experiments, with MAE values around 2×10^{-3} and RMSE around 2×10^{-2} (equivalent to 4×10^{-4} for MSE).

5.3. Research Questions

Q.1 - Is it possible to encompass more than two cities as sources in a transfer learning process?

It's indeed very much possible to encompass more than two cities as sources in a transfer learning process. Our experiments shows that using more cities can enhance the prediction accuracy as the model is able to extract more, and more diverse patterns from the data.

As we introduced on the Methodology chapter, it's indeed very possible to encompass multiple cities on the source domain and, theoretically, this can enhance the model's capabilities on generalization.

Q.2 - What's the impact of the number of source cities on the model's accuracy?

With Experiment 6 we could prove that pre-training the autoencoder in two cities instead of a single one makes a significant impact on the model's ability to reconstruct the data, implying that the features extracted when the model has access to more patterns are richer and more informative compared to the standard case.

However, this trend was not as apparent in the Full-Model Experiment, where the complete city map was utilized. The implications of this observation require further analysis on the model's architecture and the impact of the dataset distribution on the results. We conjecture that this could be due to the lack of data on the peripheral parts of the city, which doesn't add any information to the model in relation to patterns of traffic.

Q.3 - Is there a limit on the number of source cities?

Yes, there seems to be a limit on how good a model can generalize given the number (or diversity) of source domains. Experiment 2 clearly shows that the performance of an autoencoder trained with four cities exceeds the one trained with eight cities. This may be an indicative that the cities in question (or at least one of the cities in question) may not have a similar distribution as the others, which is a major assumption we made in the start of the development that allows us to use the patterns learned from one city on another.

Of course that having more data is generally accepted as the main way in which one can enhance a model, as the diversity of data is often regarded as the limit of what the model can generalize, but in order to make use of this diversity one must be sure that the distributions of the data are similar.

Despite that, one of the major limitations associated with the increase on the number of source cities is that it requires significant more computational resources. This implies that, for instance, that the model's complexity, or even architecture, needs to be adapted to enable this change, as we had to do in relation to the Batch and Layer Normalization layers.

6. Conclusion

This Chapter, as announced before, has the purpose of summarizing the thesis' key findings and outlining the contributions of the work presented here to the state-of-the-art of the traffic forecasting field, as well as showing the limitations we found during the development of these complex models and addressing the possible future path to take to overcome them. It's divided into two parts: Section 6.1 reviews the entirety of the thesis, highlighting its main points and contributions. Section 6.2 exposes the limitations faced and sets the future directions to expand upon these findings.

6.1. Summary and Contribution

As cities continue to develop into becoming *smart*, the demand for models that can predict the evolution of its data over time will only grow. As we have discussed earlier, the "cold-start problem", as the first challenge that a city needs to overcome when entering this field, will become persistent. This research aimed to propose ideas and solutions on how to tackle this issue, making *smart city* management possible as soon as sensors are installed.

The main focus of our work was on the evaluation of the impact that the diversity of domains can have on the performance of both the feature extractor as the predictor. We modeled the feature extractor as the encoder of an autoencoder. This made training and evaluating the extractor very straight-forward, as after going through the encoder and then decoder cells, the output data is supposed to be the same as the input. As a result of this convenience, we are prone to incur in two errors - one when encoding and another when decoding. Nonetheless, training the network using this schema proved to be reliable, as we could reach relatively low reconstruction errors (RMSE in the order of 10^{-2}).

Moreover, using GConvLSTM as the base cell for the autoencoder validates what was proposed on the methodology development as this architecture can effectively extract ST features from the traffic data.

The second part of the model, the predictor, was based on the A3T-GCN cell, which blends convolutional graph layers with attention, making the model capable of learning how to predict the next steps based on the complex feature tensor input.

As for the transfer learning part, the combination of pre-training with domain adversarial training yielded good results on the task. Experiments performed on the thesis confirm the hypothesis that pre-training is effective to handle lack of data on the target dataset and that the increase on the number of cities (and therefore increase on the diversity of the source domain) can also positively impact the model. Furthermore, adversarial training was proved, supported by experimental data, to be successful on generating domain-agnostic features.

Our developed model also has a remarkable capability: it can transfer learn through cities of different sizes, as the number of nodes from the graph of two distinct cities is naturally different, we are able to perform transfer learning between cities with graphs that hold distinct sizes.

As the main contribution of the thesis, the expansion of the source domain in order to enclose a higher data diversity was shown to work well in parts. Since one of the proposed experiments shows that it can lead to better generalization for the autoencoder, we can understand that there is a field to make more advancements on the prediction capabilities of transfer learning models.

On the other hand, we couldn't observe significant changes regarding the number of source cities when testing the model for the full dataset (meaning the full city map). This may imply that there is a limitation on the city map, as the excess nodes we added may not be conveying any kind of useful information (they could all be 0 if no probe cars drove on these peripheral regions), confusing the model and filling spaces on the GPU's memory.

As an variable in the full model experiments, the number of weeks available on the target dataset also didn't seem to enhance or worse the performance. This may suggest that, after pre-training the model using at least 43200 datapoints, the model can satisfactorily learn the local patterns using only 1-week worth of data of the target city (ca. 1700 datapoints).

6.2. Limitations and Future Research

During the development of the models and thesis, we've encountered some technical and methodological limitations, which show that this growing field has still much space for research and development.

One of the major limitations we incurred was limited computational capabilities, as the data and problem are inherently big and complex, we had to make adaptations to the developed model to be able to run it for the full city dataset. A clear example of this is that even thought we had access to the best available consumer-level GPU, we had to lower the batch size of the dataloaders, which led to the need of modifications on all models, as BATCHNORM is known for being unreliable or even undefined for such cases. This changes may lead to biases, as we search and select hyperparameters for a task that is not the same as the one we are testing for.

The dataset used for this work also caused us some problems. We used it as it was a very reliable dataset containing uniformly stored data from eight different cities with similar sizes, which is good for the purpose of understanding the impact of diversity of the source domain on the performance with cities that will have graphs with similar number of nodes. Our dataset, however, had problems with data availability, as most values (as shown in Section 3.1) are 0.

As we noted from the contributions, our model can learn patterns from graphs of different sizes, but it has yet to be tested whether we can have meaningful transfer between cities with very discrepant sizes.

Future developments using this work as basis can apply intra-city transfer using this model, in special with different datasets types, as some cities may have collected data on different fields that may be useful as sources. Although for this implementation to be successful, some changes in how the model handles channels may be required.

As the number of cities that collect data increase, and we have access to multiple source domain candidates, it would be interesting to have ways to better select best fit sources to adapt for a particular target. This task would involve graph analysis and classification, but can led to much more optimized models, saving computational and time resources.

Bibliography

[1] C. Yin, Z. Xiong, H. Chen, J. Wang, D. Cooper, and B. David, “A literature survey on smart cities.,” in *Sci. China Inf. Sci.*, vol. 58, pp. 1–18, 2015.

[2] J. Zhang, Y. Zheng, D. Qi, R. Li, X. Yi, and T. Li, “Predicting citywide crowd flows using deep spatio-temporal residual networks,” in *Artificial Intelligence*, vol. 259, pp. 147–166, Elsevier, 2018.

[3] W. Jin, Y. Lin, Z. Wu, and H. Wan, “Spatio-temporal recurrent convolutional networks for citywide short-term crowd flows prediction,” in *Proceedings of the 2nd International Conference on Compute and Data Analysis*, pp. 28–35, 2018.

[4] N. G. Polson and V. O. Sokolov, “Deep learning for short-term traffic flow prediction,” in *Transportation Research Part C: Emerging Technologies*, vol. 79, pp. 1–17, Elsevier, 2017.

[5] Y. Wu, H. Tan, L. Qin, B. Ran, and Z. Jiang, “A hybrid deep learning based traffic flow prediction method and its understanding,” in *Transportation Research Part C: Emerging Technologies*, vol. 90, pp. 166–180, Elsevier, 2018.

[6] Y. Liu, Z. Liu, and R. Jia, “Deeppf: A deep learning based architecture for metro passenger flow prediction,” in *Transportation Research Part C: Emerging Technologies*, vol. 101, pp. 18–34, Elsevier, 2019.

[7] D. Chai, L. Wang, and Q. Yang, “Bike flow prediction with multi-graph convolutional networks,” in *Proceedings of the 26th ACM SIGSPATIAL international conference on advances in geographic information systems*, pp. 397–400, 2018.

[8] X. Geng, Y. Li, L. Wang, L. Zhang, Q. Yang, J. Ye, and Y. Liu, “Spatiotemporal multi-graph convolution network for ride-hailing demand forecasting,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, pp. 3656–3663, 2019.

[9] B. Yu, H. Yin, and Z. Zhu, “Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting,” in *arXiv preprint arXiv:1709.04875*, 2017.

[10] Y. Huang, X. Song, Y. Zhu, S. Zhang, and J. J. Q. Yu, “Traffic prediction with transfer learning: A mutual information-based approach,” in *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, p. 8236–8252, Institute of Electrical and Electronics Engineers (IEEE), Aug. 2023.

[11] S. J. Pan and Q. Yang, "A survey on transfer learning," in *IEEE Transactions on knowledge and data engineering*, vol. 22, pp. 1345–1359, IEEE, 2009.

[12] H. Yao, Y. Liu, Y. Wei, X. Tang, and Z. Li, "Learning from multiple cities: A meta-learning approach for spatial-temporal prediction," in *The World Wide Web Conference*, ACM, May 2019.

[13] B. Lu, X. Gan, W. Zhang, H. Yao, L. Fu, and X. Wang, "Spatio-temporal graph few-shot learning with cross-city knowledge transfer," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ACM, Aug. 2022.

[14] Y. Tang, A. Qu, A. H. Chow, W. H. Lam, S. Wong, and W. Ma, "Domain adversarial spatial-temporal network: A transferable framework for short-term traffic forecasting across cities," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, CIKM '22, (New York, NY, USA), p. 1905–1915, Association for Computing Machinery, 2022.

[15] M. J. Beckmann, C. B. McGuire, and C. B. Winsten, *Studies in the Economics of Transportation*. Santa Monica, CA: RAND Corporation, 1955.

[16] H. W. Bevis, "A model for predicting urban travel patterns," in *Journal of the American Institute of Planners*, vol. 25, pp. 87–89, Informa UK Limited, May 1959.

[17] X. Ma, Z. Dai, Z. He, J. Ma, Y. Wang, and Y. Wang, "Learning traffic as images: A deep convolutional neural network for large-scale transportation network speed prediction," in *Sensors*, vol. 17, p. 818, MDPI AG, Apr. 2017.

[18] J. Lin, Y. Chen, H. Zheng, M. Ding, P. Cheng, and L. Hanzo, "A data-driven base station sleeping strategy based on traffic prediction," in *IEEE Transactions on Network Science and Engineering*, 2021.

[19] Z. Zhao, W. Chen, X. Wu, P. C. Y. Chen, and J. Liu, "LSTM network: a deep learning approach for short-term traffic forecast," in *IET Intelligent Transport Systems*, vol. 11, pp. 68–75, Institution of Engineering and Technology (IET), Feb. 2017.

[20] H. Yao, F. Wu, J. Ke, X. Tang, Y. Jia, S. Lu, P. Gong, J. Ye, and Z. Li, "Deep multi-view spatial-temporal network for taxi demand prediction," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[21] L. Wang, X. Geng, X. Ma, F. Liu, and Q. Yang, "Cross-city transfer learning for deep spatio-temporal prediction," in *Proceedings of the Twenty-Eighth International Joint Con-*

ference on Artificial Intelligence, International Joint Conferences on Artificial Intelligence Organization, Aug. 2019.

[22] S. Wang, H. Miao, J. Li, and J. Cao, “Spatio-temporal knowledge transfer for urban crowd flow prediction via deep attentive adaptation networks,” in *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, p. 4695 – 4705, 2022.

[23] X. Ouyang, Y. Yang, W. Zhou, Y. Zhang, H. Wang, and W. Huang, “Citytrans: Domain-adversarial training with knowledge transfer for spatio-temporal prediction across cities,” in *IEEE Transactions on Knowledge and Data Engineering*, p. 1–14, 2023.

[24] Y. Jin, K. Chen, and Q. Yang, “Selective cross-city transfer learning for traffic prediction via source city region re-weighting,” in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, p. 731 – 741, 2022.

[25] L. Zhang, X. Geng, Z. Qin, H. Wang, X. Wang, Y. Zhang, J. Liang, G. Wu, X. Song, and Y. Wang, “Multi-modal graph interaction for multi-graph convolution network in urban spatiotemporal forecasting,” in *Sustainability*, vol. 14, p. 12397, MDPI AG, Sept. 2022.

[26] X. Geng, Y. Li, L. Wang, L. Zhang, Q. Yang, J. Ye, and Y. Liu, “Spatiotemporal multi-graph convolution network for ride-hailing demand forecasting,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 3656–3663, Association for the Advancement of Artificial Intelligence (AAAI), July 2019.

[27] L. Wang, D. Chai, X. Liu, L. Chen, and K. Chen, “Exploring the generalizability of spatio-temporal traffic prediction: Meta-modeling and an analytic framework,” in *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, pp. 3870–3884, Institute of Electrical and Electronics Engineers (IEEE), Apr. 2023.

[28] Y. Wei, Y. Zheng, and Q. Yang, “Transfer knowledge between cities,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, Aug. 2016.

[29] S. Elmi and K.-L. Tan, “Travel time prediction in missing data areas: Feature-based transfer learning approach,” in *Proceedings - 2020 IEEE 22nd International Conference on High Performance Computing and Communications, IEEE 18th International Conference on Smart City and IEEE 6th International Conference on Data Science and Systems, HPCC-SmartCity-DSS 2020*, p. 1088 – 1095, 2020.

[30] J. Li, N. Xie, K. Zhang, F. Guo, S. Hu, and X. M. Chen, “Network-scale traffic prediction via knowledge transfer and regional mfd analysis,” in *Transportation Research Part C: Emerging Technologies*, vol. 141, 2022.

[31] Z. Ali, P. Kefalas, K. Muhammad, B. Ali, and M. Imran, “Deep learning in citation recommendation models survey,” in *Expert Systems with Applications*, vol. 162, p. 113790, Elsevier BV, Dec. 2020.

[32] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, “A kernel two-sample test,” in *J. Mach. Learn. Res.*, vol. 13, p. 723–773, JMLR.org, mar 2012.

[33] Y. Wang, Y. Chen, P. Chen, J. Hu, and H. Zheng, “Application-based cooperative content caching via tensor domain adaptation networks,” in *2022 IEEE 14th International Conference on Wireless Communications and Signal Processing, WCSP 2022*, p. 22 – 27, 2022.

[34] Y. Ganin and V. Lempitsky, “Unsupervised domain adaptation by backpropagation,” 2015.

[35] C. Eichenberger, M. Neun, H. Martin, P. Herruzo, M. Spanring, Y. Lu, S. Choi, V. Konyakhin, N. Lukashina, A. Shpilman, N. Wiedemann, M. Raubal, B. Wang, H. L. Vu, R. Mohajerpoor, C. Cai, I. Kim, L. Hermes, A. Melnik, R. Velioglu, M. Vieth, M. Schilling, A. Bojesomo, H. A. Marzouqi, P. Liatsis, J. Santokhi, D. Hillier, Y. Yang, J. Sarwar, A. Jordan, E. Hewage, D. Jonietz, F. Tang, A. Gruca, M. Kopp, D. Kreil, and S. Hochreiter, “Traffic4cast at neurips 2021 - temporal and spatial few-shot transfer learning in gridded geo-spatial processes,” in *Proceedings of the NeurIPS 2021 Competitions and Demonstrations Track* (D. Kiela, M. Ciccone, and B. Caputo, eds.), vol. 176 of *Proceedings of Machine Learning Research*, pp. 97–112, PMLR, 06–14 Dec 2022.

[36] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” in *Science*, vol. 313, p. 504–507, American Association for the Advancement of Science (AAAS), July 2006.

[37] Y. Fan, X. Yu, R. Wieser, D. Meakin, A. Shaton, J.-N. Jaubert, R. Flottemesch, M. Howell, J. Braid, L. S. Bruckman, R. French, and Y. Wu, “Spatio-temporal denoising graph autoencoders with data augmentation for photovoltaic timeseries data imputation,” 2023.

[38] M. Sabbaqi, R. Taormina, A. Hanjalic, and E. Isufi, “Graph-time convolutional autoencoders,” in *Learning on Graphs Conference*, pp. 24–1, PMLR, 2022.

[39] B. Rozemberczki, P. Scherer, Y. He, G. Panagopoulos, A. Riedel, M. Astefanoaei, O. Kiss, F. Beres, G. Lopez, N. Collignon, and R. Sarkar, “PyTorch Geometric Temporal: Spatiotemporal Signal Processing with Neural Machine Learning Models,” in *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*, p. 4564–4573, 2021.

[40] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, “Structured sequence modeling with graph convolutional recurrent networks,” in *Lecture Notes in Computer Science*, p. 362–373, Springer International Publishing, 2018.

[41] M. Defferrard, X. Bresson, and P. Vandergheynst, “Convolutional neural networks on graphs with fast localized spectral filtering,” in *Advances in Neural Information Processing Systems*, 2016.

[42] J. Zhu, Y. Song, L. Zhao, and H. Li, “A3T-GCN: attention temporal graph convolutional network for traffic forecasting,” in *CoRR*, vol. abs/2006.11583, 2020.

[43] I. Loshchilov and F. Hutter, “Fixing weight decay regularization in adam,” in *CoRR*, vol. abs/1711.05101, 2017.

[44] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, “Mixed precision training,” 2018.

[45] B. Zhou, D. He, and Z. Sun, “Traffic predictability based on arima/garch model,” in *2006 2nd Conference on Next Generation Internet Design and Engineering, 2006. NGI ’06.*, pp. 8 pp.–207, 2006.

[46] A. Palazzi and J. Liu, “Convlstm for pytorch.” `https://github.com/Jimexist/conv_lstm_pytorch`, 2020.

[47] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, “Convolutional lstm network: A machine learning approach for precipitation nowcasting,” in *Advances in neural information processing systems*, vol. 28, 2015.

[48] Y. Huang, Y. Cheng, A. Bapna, O. Firat, M. X. Chen, D. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen, “Gpipe: Efficient training of giant neural networks using pipeline parallelism,” 2019.

[49] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” 2016.