Reproductivity paper on Spatio-Temporal Self-Supervised Learning for Traffic Flow Prediction

GDL 2023
Group id: ST-SSL
Project id: 6

Akseli Suutari, Erifeoluwa Jamgbadi, Qiyang Pan, Pranav Sateesh
{akseli.suutari, erifeoluwa.jamgbadi, qiyang.pan,pranav.sateesh}@usi.ch

Abstract
The authors introduce the problem of predicting traffic flows in cities and note the challenges involved due to the heterogeneity of traffic patterns across different regions and the dynamic nature of traffic flows over time.They then propose a novel Spatio-Temporal Self-Supervised Learning (ST-SSL) framework for traffic flow prediction that uses self-supervised learning techniques to enhance the representation of both spatial and temporal heterogeneity in traffic flow data.The authors demonstrate the effectiveness of their approach through experiments on four real-world traffic flow datasets, showing that their ST-SSL framework outperforms traditional supervised learning methods and other state-of-the-art approaches for traffic flow prediction.Overall, the abstract provides a concise overview of the key contributions and results presented in this paper.

1 Introduction

The introduction of this paper provides an overview of the challenges involved in predicting traffic flows in cities and introduces the authors' proposed solution using Spatio-Temporal Self-Supervised Learning (ST-SSL).

The authors note that traditional methods for traffic flow prediction often struggle to capture the complex spatial and temporal patterns that influence traffic flows in cities. They argue that this is due to the heterogeneity of traffic patterns across different regions of the city, as well as the dynamic nature of traffic flows over time. That is one of the main reasons why graph neural network is a popular implementation to use to solve this problem.

A graph was also used to address these challenges in the authors proposed novel ST-SSL framework for traffic flow prediction. This framework uses self-supervised learning techniques to enhance the representation of both spatial and temporal heterogeneity in traffic flow data.

Specifically, their approach involves using a graph convolutional network (GCN) to encode spatial information and an adaptive graph augmentation (AGA) technique to inject spatial heterogeneity into the GCN embeddings. They also use a time-aware contrastive learning (TCL) technique to capture temporal heterogeneity.

The authors demonstrate the effectiveness of their approach through experiments on four real-world traffic flow datasets. They show that their ST-SSL framework outperforms traditional supervised learning methods and other state-of-the-art approaches for traffic flow prediction.

2 Related works

Before this paper was done, methods that were used were various machine learning and deep learning methods that were used as the architectures. Different neural networks implementation such as CNN (convolutional neural network)[1], GNN (graph neural network) [2], RNN (recurrent neural network)[3], LSTM (long short-term memory) [4] were used.

The problem with some of the methods mentioned is that some of them are good for modelling temporal features, but weak for spatial, while others may be good for spatial and weak for temporal [5]. This poses a problem in traffic prediction because for because for more accurate predictions of the traffic, both spatial and temporal is important. The location, place etc tells the place where we are basing the predictions on, while the temporal is important for telling the time when there is more or less traffic.

Some of the papers the authors looked at mentioned how GNN is a really powerful neural network that is useful for traffic predictions[6]. Some trends we noted in some papers is some variation of graph architecture was used in different papers [7]. This is mainly because of the graph structure which helps to map the relationship in the traffic data, thus helping with analyzing and making more accurate predictions.

The work presented in the paper we reproduce solves the problem of the limitations encountered using the above methods by using self-supervised learning for representation learning[8] to solve the spatio-temporal heterogeneity problem. While earlier work focused on them as individual parts, this paper solved the problem that occurred by focusing on both these important elements. By doing so, more accurate predictions can be made.

3 Methodology

This section talks about the method presented in the original paper.

3.1 Spatio-Temporal Encoder

In this section, we will explore how the Spatio-Temporal encoder is built in the paper we studied. The combination of spatial and temporal encoder is used together in this paper. Spatial deals with geographical location and area while temporal deals with the time [8], with the combination of these two to make a joint encoder, they were able to observe and keep the spatial temporal information of the traffic flow, that would otherwise be lost if not using the joint encoder.

The main component of building this encoder was the integration of two networks. The two network are temporal convolutional and graph convolutional propagation. The use of these two networks for building the encoder solved the problem of capturing the spatial and temporal patterns in the traffic flow data.[Explain what temporal convolution is ? And graph convolution propagation and its usefulness here] Improves the accuracy of the prediction.

We will divide how the encoder was built into three parts, how the spatial data was handled, how the temporal data was handled and using those separate components in building the spatio-temporal encoder.

3.1.1 Temporal For handling the temporal data, the author in the paper used a 1D casual convolution which is a neural network called CNN (convolutional neural network). A 1D CNN was used for temporal data because time is an important measurement in temporal data and it's classified as 1-dimensional data and CNN can successfully extract features of time in the temporal data [9]. How the neural network handles the structure of the data is an important factor when choosing neural networks to work with. This is one of the reason CNN was chosen, as one of the advantages of convolutions is it takes into consideration the structure of the data, which in this case is a traffic flow tensor. A gated mechanism was also used with the 1D casual convolution.
After the data was sorted, it was used as input for the temporal convolution and the output was a time-aware embedding for each region as shown in equation 1.

$$(B_{t-T_{out}}, ..., B_t) = TC(X_{t-T}, ..., X_t) \tag{1}$$

3.1.2 Spatial A graph-based structure is used for the spatial data. A graph is used because the output from the temporal encoder is a region embedding matrix at different timestep t, so using graph, we would be able to explore the relationship in the different regions, thus capturing the spatial correlations in the regions. The mechanism used is known as a graph-based message passing mechanism and that's what is used in the spatial convolution encoder.

Getting the output of the temporal encoder, which is a region embedding matrix at different timestep t and the matrix A, which is the adjacency matrix of the graph from the spatial data, it was able to be used as input for the spatial convolution encoder. The output then obtained was refined embeddings matrix of all region at timestamp t as shown in equation 2.

$$\mathbf{E}_t = SC(B_t, A) \tag{2}$$

3.1.3 Spatio-Temporal Encoder The spatio-temporal encoder is built in the following way: temporal Convolution(TC) →Spatial Convolution(SC) →the same Temporal Convolution(TC) block again. As explained for the spatial and temporal convolution in 3.3.1, the output of temporal is used as the input for the spatial convolution, then the output of from the spatial is then used as input for temporal convolution. The output each time after the final temporal convolution is a sequence of embedding matrix[8]. This is done several times (TC →SC →TC), until the temporal dimension of T' is 0 (after multiple aggregations and propagations).

This block structure is repeated multiple times to obtain a sequence of embedding matrices with a temporal dimension of T0, showing that all information has gone through the spatio-temporal encoder steps. Once this happens, the final embedding matrix is obtained, where each row is the final embedding of region rn [8]. It is now in a structure that can be used for predictions.

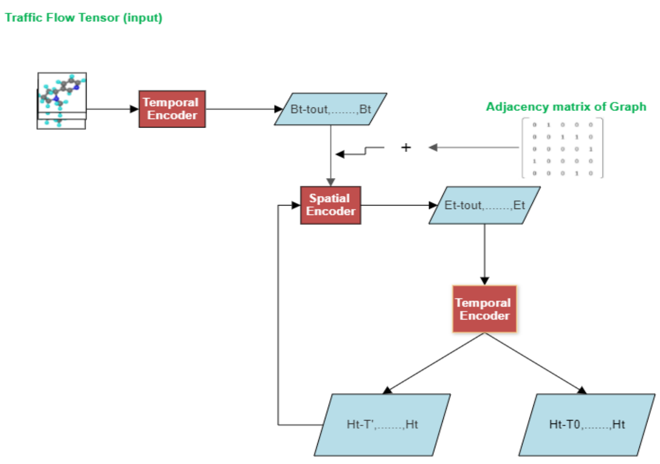


Figure 1. Spatio-Temporal Encoder Architecture

3.2 Adaptive Graph Augmentation on TFG

The Adaptive Graph Augmentation method used in the Spatio-Temporal Self-Supervised Learning (ST-SSL) framework for traffic flow prediction involves two phases of graph augmentation schemes on TFG $\mathcal{G} = (V, E, A, \mathcal{X}_{t-T:t})$ with traffic-level data augmentation and graph topology-level structure augmentation.

The paper uses a method of cosine vector similarity to estimate the heterogeneity degree between two regions over all time. Regions with smaller heterogeneity will have a higher (functional) similarity.

$$q_{m,n} = \frac{u_m^\top u_n}{\|u_m\|\|u_n\|} \tag{3}$$

where

$$u_n = \sum_{\tau=t-T}^{t} p_{\tau,n} \cdot b_{\tau,n}, p_{\tau,n} = b_{\tau,n}^{\top} \cdot w_0$$

In the first phase of graph augmentation, mask is applied to the less relevant traffic volume at τ-th time. This is done to improve the robustness of the model. Specifically, for each region r_n in the TFG, a probability $\rho_{\tau,n} \in \{0,1\}$ is generated from a Bernoulli distribution Bern(1-$p_{\tau,n}$). $\rho = 1$ indicates that the region r_n's volume $x_{\tau,n}$ at τ's time step is masked, as it's not relevant to the overall traffic volume pattern of region r_n.

In the second phase of graph topology-level augmentation, a mask is applied to remove connections between adjacent regions with heterogeneous traffic patterns and build connections between distant regions with similar latent urban function. This is done to debias region connections with low inter-correlated traffic patterns and capture long-range region dependencies with the global urban context. Specifically, a clustering-based generative task is used to learn spatial representations that can be used to predict traffic flow.

The Adaptive Graph Augmentation method is adaptive to the learned heterogeneity-aware region dependencies in terms of their traffic regularities. This allows it to capture long-range region dependencies with the global urban context and debias region connections with low inter-correlated traffic patterns. Overall, this method plays an important role in enabling adaptive spatio-temporal self-supervised learning for predicting citywide traffic flows at different time periods.

3.3 SSL for Spatial Heterogeneity Modeling

The Self-Supervised Learning for Spatial Heterogeneity Modeling method involves a self-supervised learning task that injects spatial heterogeneity into region embeddings by enforcing divergence among region-specific traffic pattern representations. Specifically, a clustering-based generative task is used to learn spatial representations that can be used to predict traffic flow.

First, they cluster the regions into K categories, each catogory represents an aggregation of city functions(residential zone, shopping mall, transportation hub) by performing $\tilde{z}_{n,k} = c_k^T \tilde{h}_n$, where $\{c_1, ..., c_K\}$ are the embeddings of the clusters and $\tilde{h}_n$ is the region encoding of region r_n after augmentation.

Second, an auxiliary learning task is designed to predict the category of each region with respect to their cluster assignment in the clustering task with the original encoding h_n of region r_n. And cross-entroy is applied as the loss function.

$$l(h_n, \tilde{z}_n) = -\sum_k \tilde{z}_{n,k} log \frac{\exp(\hat{z}_{n,k}/\gamma)}{\sum_j \exp(\hat{z}_{n,j}/\gamma)} \quad (4)$$

and

$$\mathcal{L}_s = \sum_{n=1}^{N} l(h_n, \tilde{z}_n) \quad (5)$$

However, there are still two problems with the cluster assignment matrix $\tilde{Z} = (\tilde{z}_1, .., \tilde{z}_N)$. First, The assignment of each region is not normalized so $\tilde{z}_k$ may not sum up to 1. Second, it is possible that all regions have the same assignment. The paper employs the maximum entropy principle to tackle with the problem, that is, to make sure each row of the maxtrix sums up to N/K. The optimal solution of $\tilde{Z}$ to maximize the similarity of embeddings and clusters is optimized by

$$\max_{\tilde{Z} \in \mathcal{Z}} \operatorname{tr}(\tilde{Z} C \tilde{H}^{\top}) + \epsilon H(\tilde{Z})$$

Where $H(\tilde{Z})$ is the entropy defined as $-\sum_{n,k} \tilde{z}_{n,k} log \tilde{z}_{n,k}$ and ϵ is the smoothness controller.

3.4 SSL for Temporal Heterogeneity Modeling

The authors use a SSL(Self-supervised learning) task to help them add the temporal heterogenity to the regional embeddings that are time dependant by taking the divergence of among time step-specific traffic patterns.

They add the encoded time-aware region embeddings from both the original and augmented TFGs. Given as $\boldsymbol{v}_{t,n} = \boldsymbol{w_1} \odot \boldsymbol{h}_{t,n} + \boldsymbol{w_2} \odot \tilde{\boldsymbol{h}}_{t,n}$.

By using the element-wise product where $\boldsymbol{w_1}$,$\boldsymbol{w_2}$ are the learnable parametrs for the learning task.

They then generate it for the city, including all regions for a particular time-stamp t by aggregating embeddings of all regions and call it $\boldsymbol{s}_t$.

$$\boldsymbol{s}_t = \sigma\left(\frac{1}{N}\sum_{n=1}^{N} \boldsymbol{v}_{t,n}\right) \quad (6)$$

To make the model more sensitive to the role difference in time plays such as rush hours etc., they treat the region-level and city-level embeddings $(\boldsymbol{v}_{t,n}, \boldsymbol{s}_t)$ from the same time step as the positive pairs in our SSL task, and the embeddings from different time steps as negative pairs. With this design, the positive pairs will help access information about the consistency of time-specific citywide traffic trends, while the negative pairs help in capturing the temporal heterogeneity across different time steps. Formally, the temporal heterogeneity-enhanced SSL task is optimized with the following loss with cross-entropy metric:

$$\mathcal{L}_t = -\left(\sum_{n=1}^{N} \log g\left(\boldsymbol{v}_{t,n}, \boldsymbol{s}_t\right) + \sum_{n=1}^{N} \log\left(1 - g\left(\boldsymbol{v}_{t',n}, \boldsymbol{s}_t\right)\right)\right) \quad (7)$$

where t and t' denote two different time steps. g is a criterion function defined as $g\left(\boldsymbol{v}_{t,n}, \boldsymbol{s}_t\right) = \sigma\left(\boldsymbol{v}_{t,n}^{\top} \boldsymbol{W}_3 \boldsymbol{s}_t\right)$. $\boldsymbol{W}_3 \in \mathbb{R}^{N \times N}$ is the learnable transformation matrix.

3.5 Model Training

The model is trained by feeding the embeddings $\mathbf{h}$ into a multi layer perceptron (MLP) with two fully connected hidden layers. MLP is an artificial neural network that can be trained using a technique called backpropagation. The backpropagation algorithm makes changes to the weights of perceptrons based on the loss function. **FINDREFERENCE**

The MLP predicts the traffic flow for the next time period. The prediction for the embedding $\mathbf{h_n}$ can be written as

$$\hat{x}_{t+1,n} = MLP(\mathbf{h_n}) \tag{8}$$

The loss function used for MLP is

$$\mathcal{L}_{\mathcal{MLP}} = \Sigma_{n=1}^{N} \lambda |x^{(0)}_{t+1,n} - \hat{x}^{(0)}_{t+1,n}| + (1-\lambda)|x^{(1)}_{t+1,n} - \hat{x}^{(1)}_{t+1,n}|, \tag{9}$$

Where x denotes the ground truth and $\hat{x}$ the estimation and superscripts (0) and (1) denote traffic inflow and outflow.

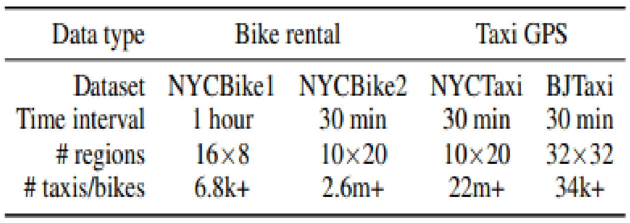
For the complete training of ST-SSL model the three loss functions are combined to one loss function $\mathcal{L}_{joint}$ that is then minimized in the training process. The loss function of the whole model is

$$\mathcal{L}_{joint} = \mathcal{L}_{MLP} + \mathcal{L}_{S} + \mathcal{L}_{T} \tag{10}$$

4 Implementation

In our study we used the author's code that they provided along with their paper. We needed to make couple minor bug fixes so that we were able to run the code on our GPU cluster.

4.1 Datasets



Data type	Bike rental		Taxi GPS	
Dataset	NYCBike1	NYCBike2	NYCTaxi	BJTaxi
Time interval	1 hour	30 min	30 min	30 min
# regions	16×8	10×20	10×20	32×32
# taxis/bikes	6.8k+	2.6m+	22m+	34k+

Figure 2. Dataset Table

There are four different datasets that was used - NYCBike1 [10], NYCBike2 [11], NYCTaxi [11] and BJTaxi [10]. The dataset was obtained from the github specified for this paper.

Each of the dataset mentioned above contains 4 files, train.npz, test.npz, val.npz and adj_mx.npz. In train.npz, test.npz, val.npz there are 4 'numpy.ndarray' objects, which is a data structure containing lists of fixed size. This data structure is used because it is convenient when using in terms of using less memory required when storage of data, so it can store large amount of data, which is useful for model training, as large amount of data is needed to train models. It is also faster to work with than using normal python list.

The train.npz is used for training the model, the test.npz is used for the testing the model after training, val.npz is used to evaluate the model when making changes to hyperparameters, before then using the testing data for final evaluation. We decided to visualise the objects in the dataset to help understand the structure of the data more. We chose the NYCBike1 train.npz dataset and visualised the 3 out of the 4 'numpy.ndarray' objects. The 4 'numpy.ndarray' objects in theses files are as follows:

- x : it is a 4D tensor that has the shape (timeslots, lookback_window, nodes, flow_types) [8] Figure 3 shows the visualisation of the x object in the NYCBike1 dataset. This shows the representation of the different flow types in the dataset for all the nodes. We found out from analysing the date that the following information: timeslots has the maximum value of 3000, there is 1 lookback_window, there are 128 node values and 2 types of flow.

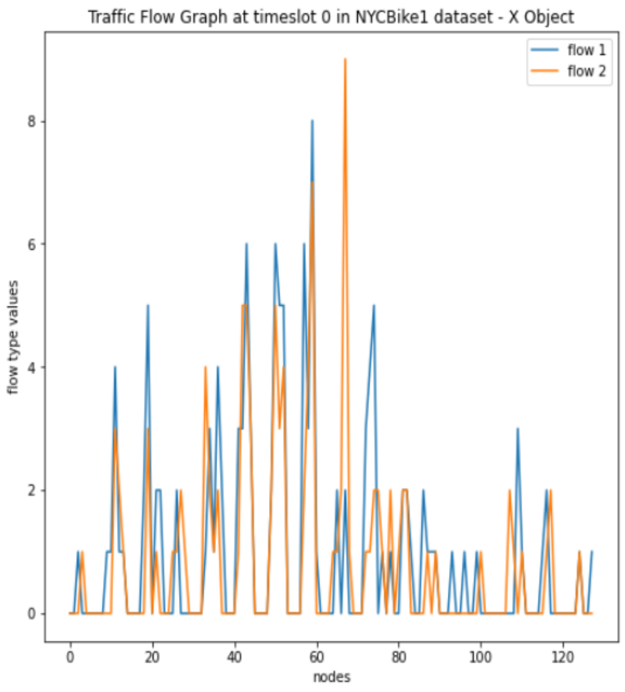


Figure 3. X object visualisation from NYCBike1 Dataset

- y: it is a 4D tensor that has the shape (timeslots, predict_horizon, nodes, flow_types). Figure 4 shows the visualisation of the y object. It has the same structure as the x object, which is timeslots has the maximum value of 3000, there is 1 lookback_window, there are 128 node values and 2 types of flow.[8]

- x_offset: shows the tensor which contains the offsets that is used for the lookback_window in the y object. Figure 5 shows the visualisation of the x_offset data. The graph shows time vs the x offset values and also that the need for offsetting the lookback window reduced as the time went on. [8]

- y_offset: shows the tensor which contains the offsets that is used for the predict_horizon in the y object. The y offset didn't plot to create any representation that was useful. The reason for this is that the yoffset only contains 1 array. Our thoughts being this is that because the predictions are made for the certain times

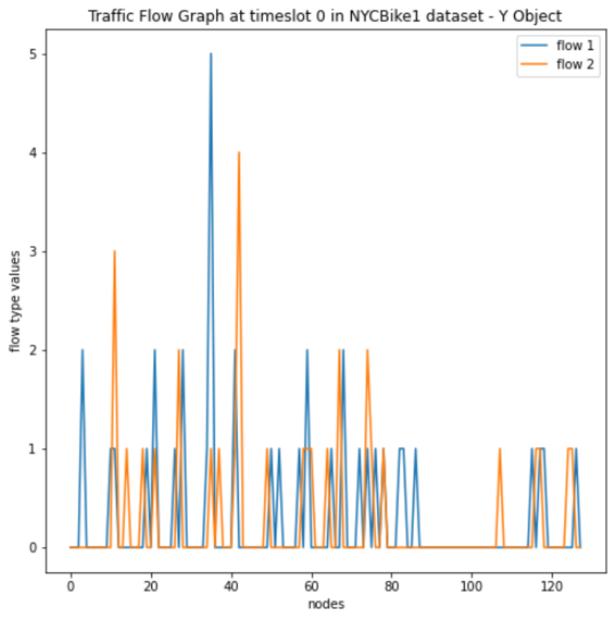


Figure 4. Y object visualisation from NYCBike1 Dataset

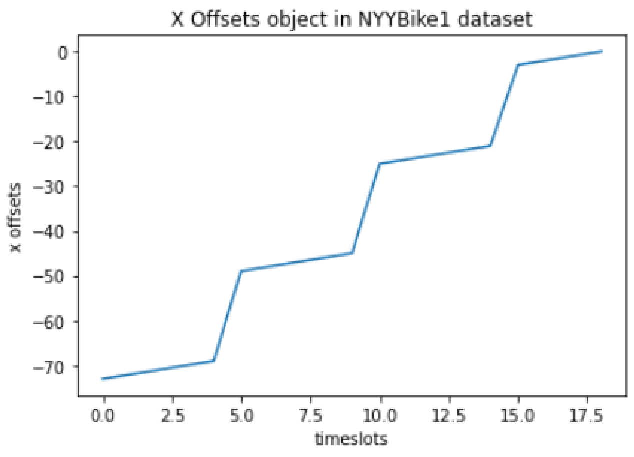


Figure 5. X offset visualisation from NYCBike1 Dataset

in which the prediction is needed for, so there is no need for any changes to the predict_horizon[8].

A timeseries visualization of the traffic inflow for the first 4 nodes can be seen in figure 6. From this we can clearly see that during the daytime the taxis are much more active than during the night time. In the morning there are also clear spikes which might be caused by people commuting to work.

The adj_mx.npz contains the adjacency matrix that will be used in the spatial encoder, as it is one of the inputs required. The value in the matrix is 0's and 1's and it is used for showing the relationship between nodes in a graph. In our case the adjacency matrix will show the relationship between the regions, which will help making traffic flow predictions in different regions.

An error we came accross in regards to the dataset information mentioned in the original paper is that they mentioned that dataset NYCBike1 [10] has a time interval of 30 minutes and NYCBike2[11]has one of 1 hour. However in their table, they say that NYCBike1 is 1 hour and NYCBike2 30 minutes interval as shown in Figure 2.

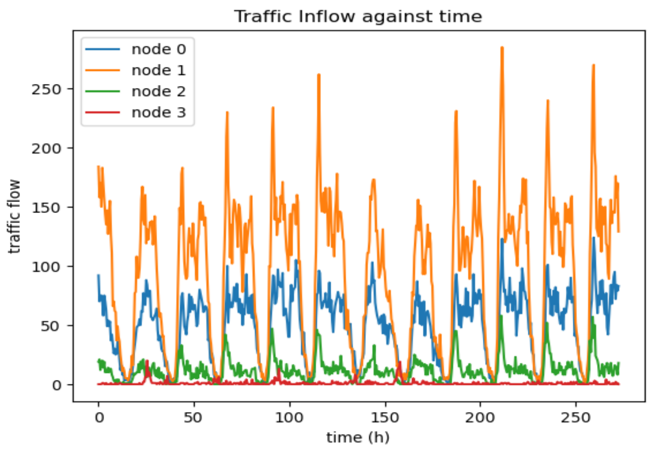


Figure 6. NYCTaxi data as a timeseries for first 4 nodes.

4.2 Code implementation

This sections goes into the technical detail of how the code was implemented[12].

The whole code and results of our experiments can be found here: `https://github.com/audioAXS/USI_GDL_ST-SSL`

4.2.1 Spatio-Temporal Encoder The code was implemented properly as described in the paper. The code implementation for the spatio-temporal encoder is divided into 3 sections, the spatio convolution layer, the temporal convolution layer and then the spatio-temporal encoder. These sections are all separate classes in the code. These classes input the nn.module, which contains all the required things needed to build and train the model in pytorch. It works by using methods in these classes such a forward method to return an output. The output for the spatio-temporal encoder is a final embedding matrix, where each row is the final embedding of region rn. The output for the temporal convolution layer is a time-aware embedding for each region, while for the spatio convolution layer, it is a refined embeddings matrix of all regions at timestamp t.

4.2.1.1. TemporalConvLayer
First it checks that the in channel and out channel is the same size using another class called align before using them as parameters to make a convolution layer. Padding is added to make the input or output depending on which is less to make them same size, thus avoiding the loss of temporal dimesntions in the temporal convolutional layer. It uses the nn.conv2d module in pytorch to make the convolution layer. It takes in parameters such as the input channel, output channel, stride of 1 and a 1x1 convolution.

If the gated linear unit function (GLU) is activated the out channel is multiplied by 2, this happens because if using GLU as an activation function, in the output of the function, the out channel is divided by 2. The default activation function that should be used is rectified linear unit (RLU), then the output channel stays as normal meaning that it is the

(RLU). The RLU is used to solve the problem of vanishing gradients in the network.

A forward method is then done in the same class. This will return the output of the temporal convolutional layer depending on which activation function is equal to self.act. The options are sigmoid or GLU.

4.2.1.2. SpatioConvLayer
The parameters are set using the pytorch module nn.Parameter e.g bias, theta. The parameters reset method resets the parameters. It uses kaiming_uniform, which is an initialization function that considers the nonlinearity of activation functions[13], which allows the complexity of the learning done by model to increase. A forward method is also implemented in this class, it takes in x tensor and the Laplacian matrix as input and returns the result calculated by the torch.relu.

4.2.1.3. STEncoder
The ST encoder contains the forward method, the Chebyshev polynomial method and a Laplacian method. The Laplacian method is used to calculate the laplacian of the graph. This shows us how well connected the graph is and this is very important as we want to understand the relationship between nodes[14]. The Laplacian method creates a 2D tensor with ones on the diagonal and zeros everywhere else. The matrix is then added to the graph and the Laplacian graph is created and returned.

The Chebyshev polynomial method is used for fast filtering[15]. and it is calculated based on the Laplacian graph created. The forward function in the STEncoder takes the input of x0 and the graph. It passes the graph to both the Laplacian method and Chebyshev polynomial method. It checks that the x is of the required size before using it as input for the temporal and spatial convolution. The x input used is of structure (batch_size, feature_dim, input_length, num_nodes). Pooling is used to keep the features of the data, while reducing the spatial dimension. Dropout is used to prevent overfitting of the data.

4.2.2 Adaptive Graph Augmentation on TFG The code defines a function called ‘sim_global‘ that calculates the global similarity of traffic flow data. The function first checks the shape of the input tensor to determine whether it represents original flow data or location embedding. If it is original flow data, the function calculates an attention scaling factor and a cosine scaling factor based on the dimensions of the tensor. It then uses ‘torch.einsum‘ to calculate the dot product between each pair of nodes in the graph.

If ‘flow_data‘ represents location embedding instead of original flow data, then only cosine scaling is calculated and used to compute pairwise similarities between nodes. Finally, the function returns a tensor representing the symmetric similarity matrix between all pairs of nodes in the graph.

The function called ‘aug_topology‘ that generates data augmentation from a topology (graph structure) perspective for an undirected graph without self-loops. It drops edges w.r.t it’s simalarity score by setting its value in the augmented adjacency matrix to 0.

The ‘aug_traffic‘ function performs data augmentation from the traffic (node attribute) perspective. It takes as input a temporal similarity matrix ‘t_sim_mx‘ and flow data ‘flow_data‘. The function calculates the probability of masking each element in the similarity matrix by subtracting it from 1 and normalizing it by its sum. It then converts this probability distribution to a numpy array. It then randomly selects elements to mask from this meshgrid using ‘np.random.choice()‘ and the probability distribution calculated earlier.

It sets these masked elements to 0 in the modified flow data using indexing. Finally, it returns the modified flow data as output.

4.2.3 SSL for Spatial Heterogeneity Modelling The module models spatial heterogeneity using a soft-clustering paradigm. The module takes as input the number of input channels ‘c_in‘, the number of prototypes ‘nmb_prototype‘, the batch size ‘batch_size‘, and a temperature parameter ‘tau‘.

It defines a linear layer called ‘prototypes‘ that maps its input to a space with dimensionality equal to the number of prototypes/clusters.

It initializes all weights in the module using a custom weight initialization function called ‘weights_init()‘.

The ‘forward()‘ method takes as input two tensors ‘z1‘ and ‘z2‘, both of shape ‘(n, l, v, c)‘ where ‘n‘ is the batch size, ‘l‘ is the sequence length, ‘v‘ is the number of nodes in the graph, and ‘c‘ is the number of input channels.

The method first computes a normalized copy of the weight tensor of the prototypes linear layer using L2 normalization. It then applies this normalized weight tensor to each input tensor after reshaping it to have shape ‘(nlv, c)‘, where ‘nlv = n * l * v‘. This produces two output tensors ‘zc1‘ and ‘zc2‘, both of shape ‘(nlv, k)‘ where ‘k‘ is the number of prototypes.

The method then applies a sinkhorn algorithm to each output tensor with detached gradients to compute soft assignments for each prototype. The resulting tensors are called ‘q1‘ and ‘q2‘.

After computing the soft assignments ‘q1‘ and ‘q2‘, the method computes two loss terms. The loss terms ‘l1‘‘l2‘ are the soft-cross-entropy loss of ‘q1‘‘q2‘ w.r.t ‘zc2‘ and ‘zc1‘.

Finally, the method returns the sum of both loss terms. This is a contrastive loss that encourages similar inputs to have similar representations in prototype space.

4.2.4 SSL for Temporal Heterogenity Modelling The model for Temporal heterogentiy uses the function *torch.nn.BCEWithLogitsLoss* to optimize, This loss combines a Sigmoid layer and the BCELoss in one single class. This is used for measuring the error of a reconstruction of the region embeddings.

4.3 Hyperparameters

For the results validation we chose to use the same hyperparameters as the authors so that we could compare the results.

We also tried different hyperparameters to see, how they affect the training accuracy and if we would get better results using those. The parameters we altered were dropout rate and augmentation percentage, which controls the adaptive graph augmentation. The different hyperparameters were studied using NYCTaxi dataset. The authors had used dropout rate of 0.1. For both hyperparameters we studied also values of 0.01 and 0.5.

Dropout is a technique that sets randomly outputs of some hidden layer neurons to zero. This is done to help the neural network to generalize better. The dropout is controlled by dropout rate hyperparameter. [16]

Table 1. Accuracy of the model using different dropout rates (average of 3 runs).

Dropout rate	Flow	MAE	MAPE
0.1	In	12.20 ± 0.03	16.81 ± 0.33
	Out	10.03 ± 0.21	17.10 ± 1.17
0.5	In	13.35 ± 0.06	18.69 ± 1.01
	Out	11.14 ± 0.24	21.57 ± 5.05
0.01	In	12.20 ± 0.03	16.81 ± 0.33
	Out	10.03 ± 0.21	17.10 ± 1.17

Table 2. Accuracy of the model using different augmentation percentages (average of 3 runs).

Augmentation percentage	Flow	MAE	MAPE
0.1	In	12.20 ± 0.03	16.81 ± 0.33
	Out	10.03 ± 0.21	17.10 ± 1.17
0.5	In	13.35 ± 0.06	18.69 ± 1.01
	Out	11.14 ± 0.24	21.57 ± 5.05
0.01	In	12.40 ± 0.40	17.35 ± 0.22
	Out	10.22 ± 0.42	18.04 ± 1.74

4.4 Experimental setup

The code was ran on a Institute of Computational Science's High-Performance Computing cluster. In the cluster GPU nodes were used to make the computations, since it is faster than using just CPUs. [17] The GPU cluster uses Slurm scheduler for the resource allocation.

The computation nodes we used had two Intel Xeon E5-2650 v3 @ 2.30GHz, 20 (2 x 10) cores CPUS and 128GB DDR4 @ 2133MHz RAM. Depending on the node that was allocated to us, the GPU used was either NVIDIA GeForce GTX 1080 Founders Edition 8GB GDDR5X with 2560 CUDA cores, NVIDIA A100-PCIe-40GB Tensor Core GPU 40GB HBM2 or two NVIDIA GeForce GTX 1080 Ti Titan 11GB GDDR5X GPUs with 3584 CUDA cores.

4.5 Computational requirements

Because we were using a computing cluster the computational settings altered based on which computing node the scheduler allocated for our computation. This made it hard to determine exact computational times, because the training times of the models altered alot. For example training times for NYCBike2 dataset different from 6.33 min up to 45.89 min when the parameters were the same and the model trained for same number of epochs.

5 Results

Table 3. Experimental results (average of 3 runs).

Methods	Flow	MAE	MAPE	MAE authors	MAPE authors
NYCBike1	In	4.98 ± 0.02	23.85 ± 0.73	4.94 ± 0.02	23.69 ± 0.11
	Out	5.34 ± 0.05	24.42 ± 0.53	5.26 ± 0.02	24.60 ± 0.27
NYCBike2	In	4.98 ± 0.02	23.85 ± 0.73	5.04 ± 0.03	22.54 ± 0.10
	Out	5.34 ± 0.05	24.42 ± 0.53	4.71 ± 0.02	21.17 ± 0.13
NYCTaxi	In	12.20 ± 0.03	16.81 ± 0.33	11.99 ± 0.12	16.38 ± 0.10
	Out	10.03 ± 0.21	17.10 ± 1.17	9.78 ± 0.09	16.86 ± 0.23
BJTaxi	In	11.36 ± 0.17	15.11 ± 0.01	11.31 ± 0.03	15.03 ± 0.13
	Out	11.44 ± 0.05	15.22 ± 0.05	11.40 ± 0.02	15.19 ± 0.15

Table 3 contains the results of our experiment. The experiment was done by training three models for all the datasets and then calculating the accuracies using the code that was provided. The models were trained such that different random seeds were used to make the resulted accuracies as generalized as possible.

Our results align quite well with the results in the original paper. Only in dataset $NYCBike2$ our MAPE in Out flow was 24.42 and the authors had presented a MAPE of 21.17 ± 0.13. This means that our results don't match the original paper.

The results of the study of dropout rate and augmentation percentage can be seen from tables 1 and 2. As we can see, changing the parameters from the baseline 0.1 made the performance of the model worse. The authors did not state in their study, how they determined the used hyperparameters. Because training these models are computationally quite heavy, it might not be possible to use modern hyperparameter tuning techniques such as machine learning to learn the optimal parameter values. Instead it might be more practical to just make initial guesses and then tune them with trial and error.

6 Discussion and conclusion

We reproduced and analysed the code from the paper assigned, which is based on a new framework that implemented spatio-temporal self-supervised learning. In our The reproductivity of the paper was a success. They had a bit of a different result compared to our result, but it the difference wasn't that large. Onlt the outflow for the NYCBike2 dataset, we got worst result, but we believe it might have been due to some computational stuff from our part. Another reason we think why the result was worst for that dataset is because, We only did the average of 3 runs and they might have done more runs, so therefore achieved better results. Apart from that, everything else looks similar to the results they achieved in the paper.

Although, we were able to successfully run the code, we had to make a few changes to the code for that to happen. We needed to make a change in aug.py file. We added .cpu() to the drop_prob, so that we could remove the error, which is as follows: "can't convert cuda:0 device type tensor to numpy. Use Tensor.cpu() to copy the tensor to host memory first". What we did we the change we made by adding the .cpu() is that, it takes the matrix of the tensor from gpu memory to cpu.

This method implemented by the authors of the paper we conducted reproductivity showed the importance of a self-supervised learning spatial and temporal data paradigm for making traffic flow predictions. This effectively removed the spatio-temporal heterogeneity problem, which was encountered using traditional methods.

References

[1] Dekang Qi Junbo Zhang, Yu Zheng. Deep Spatio-Temporal Residual Networks for Citywide Crowd Flows Prediction. https://arxiv.org/abs/1610.00081,. [Online; accessed 17.05.2023].

[2] Chao Huang Lianghao Xia Xiyue Zhang, Yong Xu. Spatial-Temporal Convolutional Graph Attention Networks for Citywide Traffic Flow Forecasting: Proceedings of the 29th ACM International Conference on Information Knowledge Management. https://dl.acm.org/doi/abs/10.1145/3340531.3411941. [Online; accessed 17.05.2023].

[3] Empowering A* Search Algorithms with Neural Networks for Personalized Route Recommendation. Unsupervised Anomaly Detection of Industrial Robots Using Sliding-Window Convolutional Variational Autoencoder. https://arxiv.org/abs/1907.08489. [Online; accessed 17.05.2023].

[4] Yu Yang Xianglong Luo, Danyang Li and Shengrui Zhang. Spatiotemporal Traffic Flow Prediction with KNN and LSTM. https://www.hindawi.com/journals/jat/2019/4145353/. [Online; accessed 18.05.2023].

[5] Jianjiang Chen Dongjin Yu Dongjing Wang Bao Li, Quan Yang and Feng Wan. A Dynamic Spatio-Temporal Deep Learning Model for Lane-Level Traffic Prediction. https://www.hindawi.com/journals/jat/2023/3208535/#related-work/. [Online; accessed 18.05.2023].

[6] Miao He Weiwei Jiang, Jiayun Luo and Weixi Gu. Graph Neural Network for Traffic Forecasting: The Research Progress. https://www.mdpi.com/2220-9964/12/3/100/. [Online; accessed 18.05.2023].

[7] Jianfeng Ma Linjie Zhang. A Spatiotemporal Graph Wavelet Neural Network for Traffic Flow Prediction. https://www.sciencedirect.com/science/article/pii/S2949715923000021. [Online; accessed 17.05.2023].

[8] Jiahao Ji, Jingyuan Wang, Chao Huang, Junjie Wu, Boren Xu, Zhenhe Wu, Junbo Zhang, and Yu Zheng. Spatio-temporal self-supervised learning for traffic flow prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.

[9] Bizhong Xia Wei Wang yongzhi Lai Tingting Chen, Xueping Liu. Unsupervised Anomaly Detection of Industrial Robots Using Sliding-Window Convolutional Variational Autoencoder. https://www.researchgate.net/publication/339664991_Unsupervised_Anomaly_Detection_of_Industrial_Robots_Using_Sliding-Window_Convolutional_Variational_Autoencoder. [Online; accessed 17.05.2023].

[10] Dekang Qi Junbo Zhang, Yu Zheng. Deep Spatio-Temporal Residual Networks for Citywide Crowd Flows Prediction. https://arxiv.org/abs/1610.00081,. [Online; accessed 18.05.2023].

[11] Hua Wei Guanjie Zheng Zhenhui Li Huaxiu Yao, Xianfeng Tang. Revisiting Spatial-Temporal Similarity: A Deep Learning Framework for Traffic Prediction. https://arxiv.org/abs/1803.01254. [Online; accessed 18.05.2023].

[12] Chao Huang Junjie Wu Boren Xu Zhenhe Wu Junbo Zhang Yu Zheng Jiahao Ji, Jingyuan Wang. ST-SSL/model at main · Echo-Ji/ST-SSL. https://github.com/Echo-Ji/ST-SSL/tree/main/model. [Online; accessed 18.05.2023].

[13] Shaoqing Ren Jian Sun Kaiming He, Xiangyu Zhang. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. https://arxiv.org/abs/1502.01852v1. [Online; accessed 18.05.2023].

[14] John D. Cook. Measuring connectivity with graph Laplacian eigenvalues. https://www.johndcook.com/blog/2016/01/07/connectivity-graph-laplacian/#:~:text=Spectral%20graph%20theory%2C%20looking%20at,A%20is%20the%20adjacency%20matrix. [Online; accessed 18.05.2023].

[15] Pierre Vandergheynst Michaël Defferrard, Xavier Bresson. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. https://proceedings.neurips.

cc/paper_files/paper/2016/file/04df4d434d481c5bb723be1b6df1ee65-Paper.pdf. [Online; accessed 18.05.2023].

[16] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[17] Institute of Computational Science. HPC. https://intranet.ics.usi.ch/HPC. [Online; accessed 04.05.2023].