Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Exploring Graph Neural Networks and Hierarchical Structures for Traffic Forecasting

Bachelor's Thesis

Justas Katkus

jkatkus@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Florian Grötschla, Joël Mathys
Prof. Dr. Roger Wattenhofer

March 17, 2023

Acknowledgements

I would like to thank Florian Grötschla and Joël Mathys for supervising this thesis and helping me whenever I had any difficulties. Moreover, I want to thank Professor Roger Wattenhofer and the Distributed Computing Group for providing me with the opportunity to work on this interesting topic.

Abstract

Traffic forecasting plays a crucial role in managing traffic systems and improving transportation efficiency [1]. Traffic4Cast is an annual competition that aims at using artificial intelligence to improve traffic predictions based on real-world data [2]. In recent years, the use of Graph Neural Networks (GNNs) has shown great potential to reliably predict traffic flow as they are especially useful when operating on graph-structured data. As the input to the challenge is built upon the road graphs of three cities, we utilize the advantages of a GNN to approach the competition task. We experiment with different features and additional data, to see how this affects the quality of the predictions. Further, we extend the hierarchical approach of one of the submissions to the Traffic4Cast competition [3] to investigate if altering the graph structure can improve the predictions. Moreover, we analyze the model architecture of this submission to make further adjustments.

Contents

Chapter 1

Introduction

Fast and reliable transportation is one of the most important features of a modern city's infrastructure. Traffic congestion not only makes the transportation experience worse but also has a negative economic and environmental impact. To tackle this issue, traffic forecasting is one of the main methods for improving transportation. Traffic forecasting has been an important subject in transportation research for over four decades [4] and has consistently improved with new techniques and more accurate data.

Traffic forecasting is a complex problem as it requires the consideration of multiple variables and uncertainties. When predicting traffic, we have to be aware that different external factors like the weather, special events, or accidents can influence the traffic flow to a great extent. The number of factors to consider and the large road graphs makes traffic forecasting computationally intensive.

Traffic4Cast is a competition organized by the Institute of Advanced Research in Artificial Intelligence (IARAI) and is about the application of AI to forecast traffic. This competition is taking place every year with a different setting - In 2022 the goal of the competition is to predict congestion classes for entire road graphs 15 minutes into the future by using car count data from spatially sparse loop counters. This data is given for three cities (London, Madrid, and Melbourne) for which we need to make predictions.

To tackle this challenge, we use a Graph Neural Network (GNN). GNNs are a family of neural networks that can operate naturally on graph-structured data [5]. As the data of the competition is given as nodes and edges from OpenStreetMaps [6], the easiest representation of the road map is a graph. With that in mind, it makes sense to propagate information through the graph by using a GNN.

As a baseline solution to this problem, we use the approach and code of my supervisors [3] which is also a submission to the competition. After exploring the given approach and data, the goal is to further improve the solution.

A neural network can not learn to make the right predictions without having the right input. Therefore we experiment with additional features to see if they could improve the quality of the predictions.

Further, we work on the structure of the road graph for more insights into how the information gets propagated. For that, we implement a hierarchical structure on top of the existing graph for a wider distribution of the input data. Moreover, we analyze the existing architecture of the model to further improve it. For each of these approaches, we run experiments to verify what influence they have on the predictions.

Chapter 2

Problem description

The Traffic4Cast challenge provides us with the data for 3 cities - London, Madrid, and Melbourne. For each of these cities, we get the complete road graphs as a static parquet file which we can use to build our graph for the GNN.

As input for the predictions, we get car counter data for some nodes of the graph. The data comes from traffic loop counters which are placed only at certain points in the cities, which is the reason why the input data is spatially sparse. The traffic counter values are represented as nodes with measured volumes per 15-minute time bin.

In the challenge, the goal is to predict the congestion classes of the entire road graph based on the traffic counter data of the last hour, which would include up to 4 traffic counter values for each node as the traffic counter measurements happen every 15 minutes.

Another part of the given data is the ground truth labels. These are generated with the use of aggregated GPS probe data and are again spatially sparse. For this challenge, there are only 3 congestion classes, which means the labels can only have 3 possible values. These 3 congestion classes can be understood as the red, yellow, and green coloring of common traffic maps.

Figure 2.1: This Figure shows the ground truth labels for a part of London at a specific time point. Adapted from [2].

Every city contains between around 49500 and 63400 nodes with a little bit less than 4000 counter nodes. For each of the 3 cities, we get the input data for a time frame of approximately 6 months.

The goal of the Traffic4Cast challenge is to train a model for each of these cities which would predict congestion classes for the entire road graph 15 minutes into the future [7]. As we already have a baseline solution [3] to this problem, our goal is to improve it. The quality of the predictions is evaluated by using a masked cross-entropy loss on the congestion classes [2].

Chapter 3

Preliminaries

3.0.1 GNNs

GNNs (Graph neural networks) GNNs are a family of neural networks which work well on graph-structured data [5]. To further investigate the concept of GNNs we need to define what a graph is. A graph represents relations (edges) between a set of points (nodes). In this particular example of the Traffic4Cast challenge, we treat the road map as a graph. The nodes would be defined as points in the road map where two nodes would be connected by an edge whenever there is a direct connection (not going over any other nodes) between the two nodes.

The problem with graph-structured data in machine learning is that they do not have a fixed form and have a large dimensionality which makes it difficult to work on graphs with conventional machine learning algorithms. This is where the GNNs come in as they can operate directly on a graph.

A GNN uses multiple rounds of message passing to get information across a graph. In every round, each node computes its new state by using the current states of all its neighbors and itself. Therefore, information travels only one hop per round. Moreover, we define the state s of node v at time step t in a graph $G = (V, E)$ as $s_v^t = AGGR(\{s_u^{t-1} \mid (u, v) \in E\} \cup \{s_v^{t-1}\})$ where $AGGR$ can be any permutation-invariant function [8]. We iterate over multiple rounds of message passing to arrive at a final state of the graph.

3.0.2 PCA

Principal component analysis (PCA) is a common technique for reducing the dimensionality of a dataset while maintaining as much information as possible. PCA is useful for reducing a large dataset into single principal components. We will use this technique to reduce a dataset of past traffic counters into one variable per node at a given time.

Let's say our dataset can be represented by a matrix A. Then, to apply the PCA we first need to compute the covariance matrix of A. The covariance of two variables X and Y is defined as $cov(X,Y) = \frac{1}{n}\sum_{i=1}^{n}((x_i - \overline{X}) * (y_i * \overline{Y})$ where x_i and y_i are the members of X and Y respectively, n the number of members and $\overline{X}, \overline{Y}$ the means of all X and Y variables [9]. We can then build the covariance matrix A_{cov} by computing the covariance for all pairs of columns in the original matrix A.

For the PCA, we then need to compute the Eigenvectors and Eigenvalues of A_{cov}. After sorting the Eigenvectors by decreasing Eigenvalue, the first k Eigenvectors correspond to the first k principal components. By reorienting the axes of the data in A to the principal components we arrive at the result of PCA with which we have reduced the dimensionality of the dataset [10].

Chapter 4

Related work

4.1 Traffic forecasting

Traffic forecasting has been important already for a long time [11]. There are multiple different approaches to tackle this task. In general, one can put all approaches into two categories of methods: the knowledge-driven approach where we build our predictions using equations, or other methods requiring the exact knowledge of how the transportation system works. The other category of methods is the data-driven approach in which we use real data to predict traffic congestion [12]. Our approach with the GNN falls into the data-driven category.

The use of GNNs for traffic predictions is a relatively recent development, which improves the data-driven approach. Previous work [12] compares the performance of a GNN for traffic predictions with state-of-the-art baselines at that time. As a result, they observe a significant improvement in performance when using a GNN.

When applying the data-driven approach, we use in general the fact that most of the cars that will be on an edge soon are already somewhere on the road graph [13]. Having that in mind, we can predict the short-term future traffic congestion by forecasting the traffic flow.

An important part of working on traffic forecasting is the feature selection for the model. A good selection of features can significantly improve the forecasting accuracy [14]. Some solutions like the one implemented in [15] even use a separate algorithm to determine the important features which should be included in the input of a model.

4.2 Traffic4Cast competition

Traffic4Cast is a yearly competition from the institute of advanced research in artificial intelligence (IARAI) which explores traffic forecasting. As this thesis

builds upon the Traffic4Cast challenge from 2022 we can look into other submissions to this competition.

The submission with the best results [16] not only uses multiple GNNs but also reconstructs the missing input data by using a Transposed Variational Autoencoder. Another approach that made this submission so successful is splitting the dataset into five subsets, training a model on each of the subsets, and taking the average of the five predictions from the models.

Another interesting submission ended up in second place in the challenge [17]. They are using a PCA on the dataset to create additional features for the network. This should help include the global city traffic context into the model. We will as well use a PCA, but with a slightly different approach.

4.3 Hierarchical GNNs

In one of the experiments conducted, we analyze if changing the structure of the graph by implementing a hierarchical GNN would improve the predictions. Hierarchical GNNs can help get information across a large graph by converting a flat graph into a super graph which consists of multiple layers of nodes [18]. Other work with hierarchical GNNs [19] [18] show the potential of a hierarchical graph structure for different problems.

Chapter 5

Baseline architecture

As noted before, we use the model from one of the submissions to the competition [3] as a baseline solution for this challenge. To further improve it we first need to have a broad overview of how this model is constructed and what it already contains.

5.1 Model

5.1.1 Overview

First, the input graph gets constructed by using the road graph of the cities and adding the features to the nodes and edges. The GNN performs six rounds of message-passing graph convolutions. The first four rounds operate on the original road graph whereas the last two rounds also include the extended version of the road graph by using the supersegments as explained in 5.1.2. In the next chapters, we will refer to the number of graph convolution rounds as the number of GNN layers.

After the state of all nodes and edges has been computed by the GNN we use a predictor network with which we can predict the congestion classes for every edge in the graph.

5.1.2 Supersegments

The supersegments are precomputed and given as additional input to the challenge. Each supersegment represents the shortest path between two key intersections and therefore is a list of nodes. By using the supersegments we can connect important nodes in the graph and improve the flow of information with that. The mentioned submission [3] experiments with different approaches to represent the supersegments, here we will explain and use two of them.

In the first approach, we add a new node to the graph for each supersegment. We

connect this newly constructed node to all the nodes which are contained in the supersegment. For the second approach we do the same but additionally connect the two key intersections for each supersegment. We define the second approach as our baseline solution, will however use the first approach in some of the runs.

5.2 Features

In this architecture both nodes and edges have features. Each node in the graph has 14 features. These include the volumes from the loop counters for the past hour, which are represented by 4 features as the measurements are made in 15-minute intervals. Another 8 features which we will further call the "global features" contain the mean and the standard deviation of the traffic counter values of the complete graph for each 4 of the 15-minute intervals. These features are added to give a more global view of the current traffic state. The last 2 features of the nodes contain the x and y GPS coordinates of the node normalized to the interval of $[0, 1]$.

Each edge of the road graph has 4 features. These features include the normalized OpenStreetMaps values of parsed_maxspeed, importance, and length. The fourth feature contains the length of the edge divided by the parsed_maxspeed which represents the time it takes to travel through the edge. The added edges for the supersegments as explained in 5.1.2 do not have any edge features.

5.3 Dataset

We always use the complete training dataset as an early experiment showed that the performance decreases significantly when using fewer data. For each of the runs and experiments, we use a training/validation set split of 80%/20% for the training of the model.

The dataset consists of 6 months of data for each city. As every second week is used for the test dataset this leaves us with approximately 90 days of training data. The data for each day is continuous and contains 96 entries. For each of these entries, we get the measured traffic counter volumes for some nodes and the corresponding edge labels.

Chapter 6

Approaches

We use three different approaches to this challenge. We add new features to the nodes of the graph to have more information contained in the network. This should help the training of the network if the new features correlate with the predictions it has to make. We make changes in the graph representation to experiment if we can improve the message passing over the graph. We also make some changes in the architecture of the model to find the best configuration for this task.

6.1 Feature engineering

6.1.1 Global features

Approach

Global features are added to the model to include a broader view of the current traffic state. These global features are defined as the mean of all traffic counter values in the graph at the current moment and the standard deviation of it. That results in 8 additional features per node as we are considering the data of the last hour, which is split into 15-minute intervals and therefore contains 4 data points.

It is unclear how large the impact of these features is as they could be learned from the individual values of all nodes. Therefore we conduct this specific experiment to see the impact of the global features on the quality of the predictions. That would help us to see if it is possible to further improve the model by adding other global features.

Results

We use the model described in Chapter 5 to generate a baseline run. To see the effect on the performance of the global features we then use the same model

without the 8 global features. The loss curves for these runs can be seen in Figure 6.1. Note, that only the runs for London are displayed.

The results here show, that the global features are important as the training loss increases significantly when removing these features from the input. This indicates, that it might be interesting to experiment further with other additional features that give a broader view of the current state of the graph.

Run	Epoch	Training loss
Baseline	50	0.8703
Without global features	50	0.8876

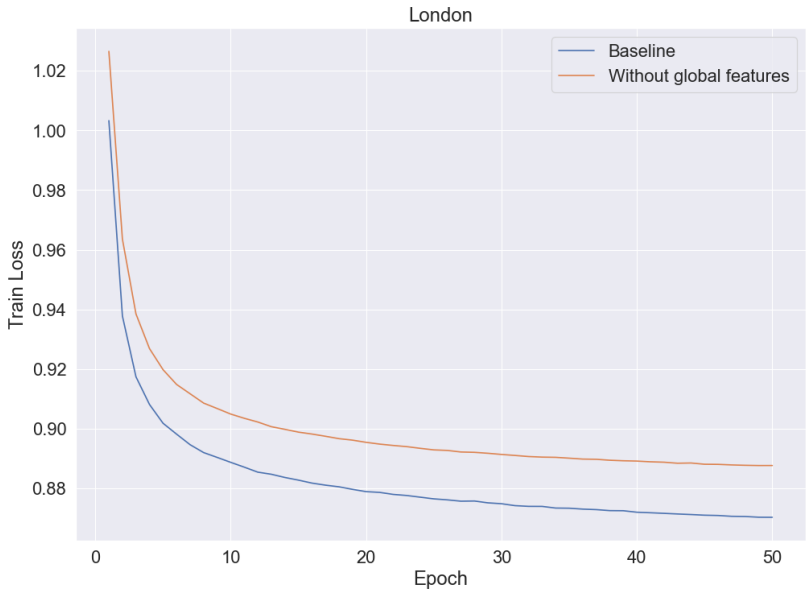

Figure 6.1: Training scores for this approach

6.1.2 Airport data

Approach

Especially in larger cities it often happens that the traffic volume increases towards the bigger airports and behaves differently than in other areas of the city. Including the distance from a node to the nearest airport in the features of the GNN would give this information to the network.

We use a public database of airports [20] which includes the coordinates of most

airports in the world. With these coordinates, we compute the euclidean air distance to the nearest airport for each node in the graph. As a next step, we map the distance values to the range of [0,1]. For that, we define for each city the value to be 0 for the node which has the smallest distance to an airport and 1 for the node which has the greatest distance to an airport. Every node between is then mapped linearly to the range [0,1] according to the distance values from the step before.

Results

The baseline run here is again the model described in Chapter 5. For the run with the additional airport data, we insert the distance value as computed before as an additional node feature. Apart from that change, both models are equal. We can see from the results that the performance indeed improved just by adding this new feature which is in line with our assumption.

Run	Epoch	Training loss
Baseline	50	0.8703
With airport data	50	0.8678

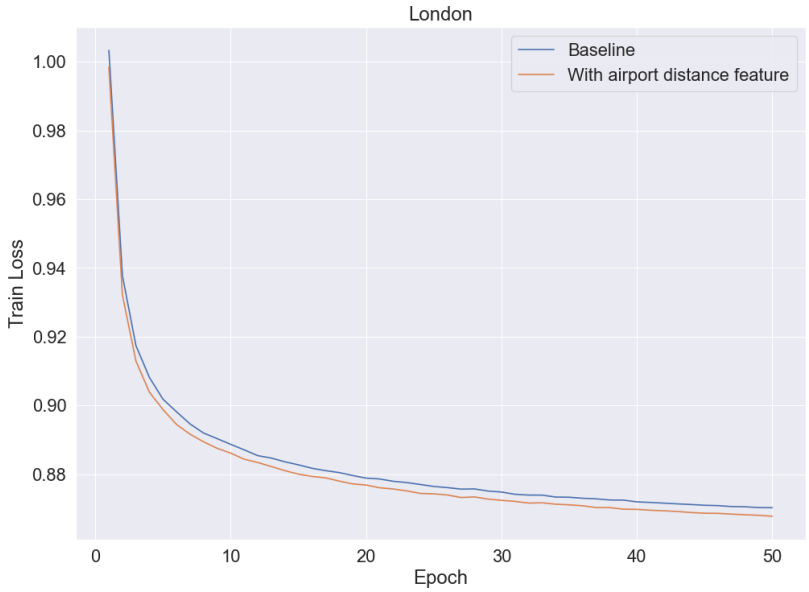

Figure 6.2: Training scores for this approach

6.1.3 Node embedding

Approach

Besides the traffic counter values and the global features, only the GPS coordinates of the nodes are included in the features of the baseline model. These coordinate features help the network to make use of the fact that close nodes tend to have similar traffic congestion states. However, there are no other relations between the nodes that can be learned from the features.

To address this, it can be useful to include an embedding of the nodes as a feature as this would allow the network to learn these relationships. We build this embedding upon the node indices with the assumption that nodes with similar indices will have something in common.

This embedding is initialized by mapping the OpenStreetMaps node indices (node_ids) given by the Traffic4Cast challenge to the range of [0,1,2, ... , length(node_ids)] to avoid using the vastly big numbers of the node indices. We use the embedding method from PyTorch to create the embedding.

Results

The baseline run here is again the model described in Chapter 5. The only change of the run with the node embeddings is the additional node feature of the node embedding as explained before. The results show, that the node embedding improved the predictions. However, the improvement is not as significant as for other approaches.

Run	Epoch	Training loss
Baseline	50	0.8703
With node embedding	50	0.8688

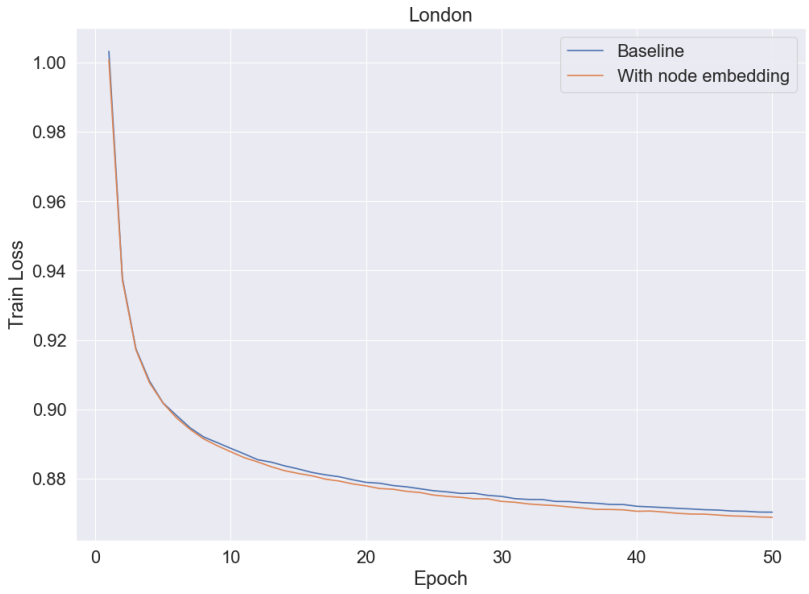

Figure 6.3: Training scores for this approach

6.1.4 PCA

Approach

Another approach is to use a PCA to include more traffic counter values in the input as mentioned in Chapter 4. More precisely we split a calendar week into 2-hour intervals resulting in 84 different time intervals. For each of those, we select all the traffic counter values recorded during that time over the complete data set. To reduce the dimension of this data we perform a PCA on the resulting $n \times k$ matrix with $n = 84 \cdot$ number of nodes in the graph and $k = 8 \cdot$ number of weeks in the data set.

For that, we fit the PCA on the complete data set as constructed before and then apply it directly to the same data set. After the computation of the PCA, we include the first 4 principal components for each node and time interval into the input of the challenge. The PCA values are stored in a separate CSV file and are included in the graph during its construction.

These additional features contain information about the traffic situation at that time in the past. We make the split over a week as the traffic is significantly different depending not only on the time but also on the weekday. By including the PCA values in the features of the GNN the model can make predictions not only based on the present traffic counter values but also considering the past.

The explained variance is a statistical measure of how much variance in a dataset can be attributed to each of the principal components [21]. When looking at the relationship between the explained variance and the number of principal components in Figure 6.4 we can see that the first 4 components already explain around 83% of the variance.

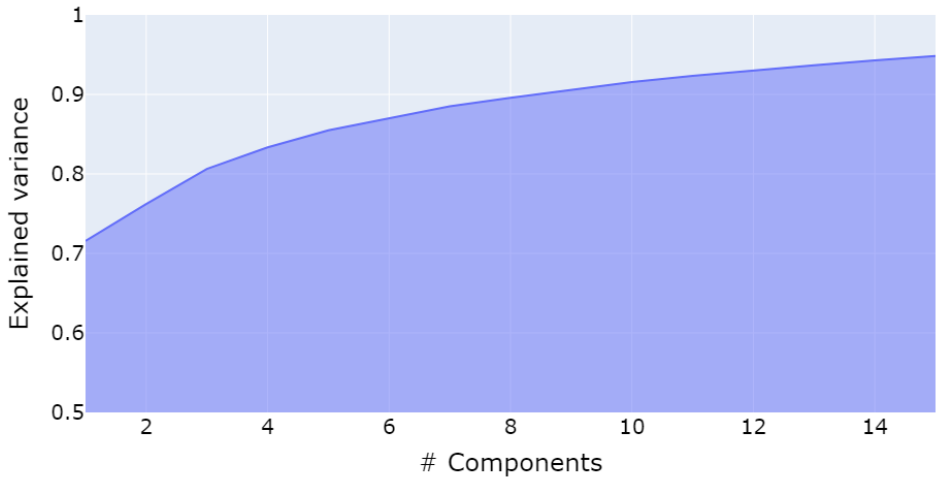

Figure 6.4: The explained variance of the principal components shows how much variance of the complete dataset can be explained by each of the principal components of a PCA. Here we can see the explained variance of each of the components for our approach explained in 6.1.4.

Results

The baseline run is made in this case with the model described in 5 with a few changes. We use the first supersegments approach for this baseline as it proved to perform better [3]. As a second change, the embedding size of the GNN is adapted from 256 to 384.

The run with the PCA features has the same properties as the defined baseline except having the 4 PCA values computed before as additional node features.

We can see, that the additional features increase the performance of the network. However, the effect is not as great as we expected it to be. This indicates that the current traffic state already contains the information needed for the predictions and that the past traffic counter values are not necessarily needed.

Run	Epoch	Training loss
Baseline	50	0.8602
With PCA features	50	0.8587

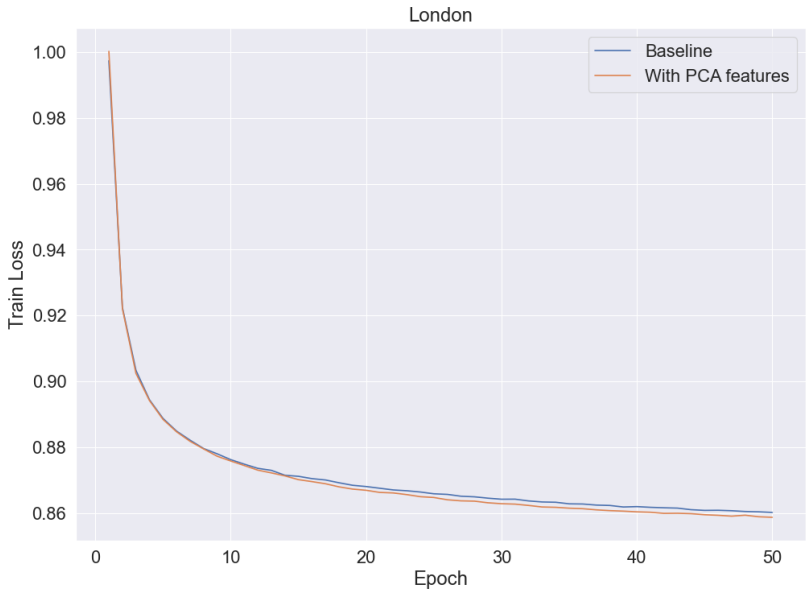

Figure 6.5: Training scores for this approach

6.2 Clustering

Message passing is a local process that only passes the information to adjacent nodes. With the size of the graph and the sparsity of the input data, this process can take a long time until it passes the information through the complete graph. To address this issue we can look at graph augmentations which could help spread the information more quickly. We test this hypothesis by using two different approaches.

6.2.1 Supersegments

Approach

One part of the input and the baseline architecture is the supersegments. These are given as additional input to the challenge. They are precomputed by choosing approximately 400 key intersections, sampling the 10 nearest neighbors at a distance greater than 500 meters, and routing between those key intersections to obtain supersegments between pairs of key intersections [22].

We want to measure the impact and performance of the supersegments by replacing the given supersegments with other nodes. As noted before, the input for the network comes mainly from traffic counters which are only placed at certain nodes. These nodes contain the most information about the current traffic state and therefore we want to use those as our most important nodes. We specifically choose the traffic counter nodes so we can measure if the given intersections are impactful. We apply the same computation as in the generation of the original supersegments to the chosen traffic counter nodes.

For that, we first sampled at random approximately 500 traffic counter nodes. The selected nodes for London can be seen in Figure 6.6. As a next step, we compute for each of these nodes their 10 nearest neighbors in the graph with a travel distance larger than 500 meters. Further, we compute the shortest paths between each node and its 10 nearest neighbors. To compute that we use the Dijkstra algorithm. As the edge weights here we use the approximate travel time through the edges computed as the length of the edge divided by the speed limit of the edge. This procedure leaves us with around 5000 supersegments with which we replace the original ones for comparison.

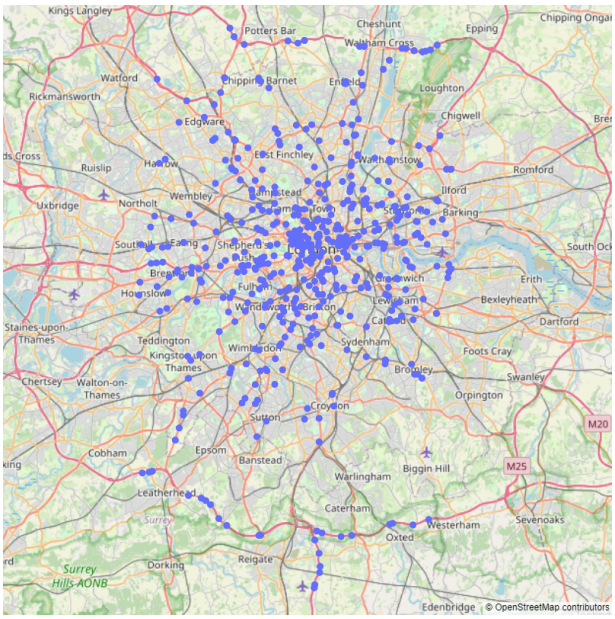

Figure 6.6: Distribution of the 500 selected traffic counter nodes in London for the computation of the supersegments

Another comparison we want to make is to find out what would happen if we would run the same process on arbitrary nodes. For that, we sample 500 nodes

from the road graph at random and again compute the supersegments between those as described before.

But do the segments have to be connected paths to increase the network performance? To answer this question, we prepare 2 different sets of nodes that are computed differently than before. The first approach samples 10 nodes at random for each of the around 5000 supersegments and replaces the original supersegments with these sets of nodes. In the second approach, we sample 5000 nodes at random as a first step. For each of these nodes, we then compute their 10 nearest neighbors by euclidian distance to then include these nodes as the supersegments into the input of our network.

Results

We test the different supersegment configurations as described before by simply replacing the given supersegments file from the Traffic4Cast challenge with our new supersegments. The differences to the original model are that we use the supersegments approach 1 instead of approach 2 and we include the PCA and airport distance features from before.

As the baseline, we use the original supersegments. We run the same model for all the created supersegments as explained in 6.2.1. We can see that the changing of the supersegments does not have a great impact on the performance.

An interesting observation is that the run where we compute the supersegments based on random traffic counter nodes is the run with the greatest training loss. The run with the lowest training loss uses supersegments that are computed based on random nodes and performs even slightly better than a run with the original supersegments. One reason for that could be, that if we build the supersegments only on top of the nodes that already contain traffic counter values, the nodes that do not have any input (as there is no traffic counter at that nide) get less information than the nodes that already have an input.

Run	Epoch	Training loss
Random nodes into segments	49	0.8658
Original segments	49	0.8665
Random nodes into neighbourhood	49	0.8670
Random nodes	49	0.8671
Random traffic counter nodes into segments	49	0.8699

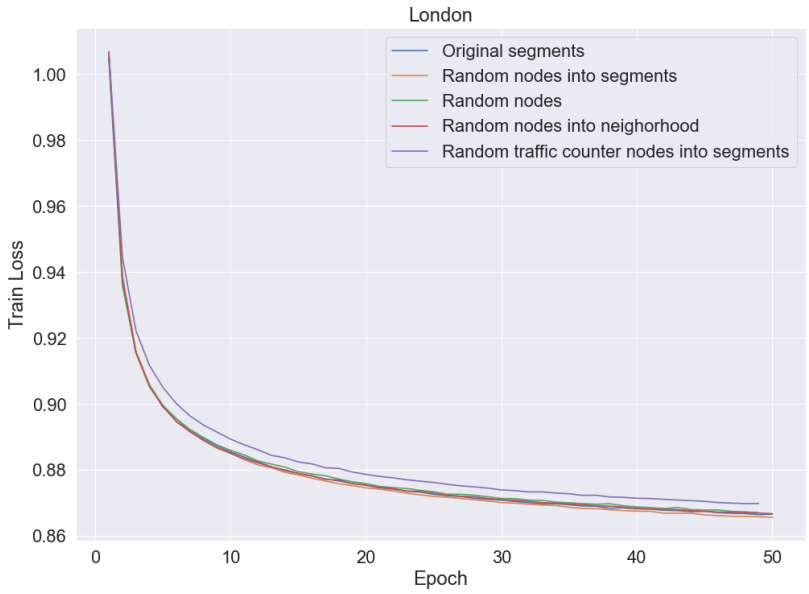

Figure 6.7: Training scores for this approach

6.2.2 Superclusters

Approach

With the supersegments we experiment with what impact the replacement of the already existing supersegments would have on the performance of the network. With the superclusters, we want to extend the graph structure by adding new layers of nodes to it.

The idea is to improve the distribution of information within the graph by implementing a hierarchical clustering with multiple layers. First, we need to define what a cluster and a partition are. We define a cluster as a set of nodes. Let's say n_{set} is a set of nodes, then we define a partition $P(n_{set})$ as a set of clusters such that each node in n_{set} is contained in one and only one cluster within $P(n_{set})$.

We want to implement multiple layers of partitions with the original graph nodes being on the lowest level and each layer would represent a partition. In the levels above the original graph nodes, each (artificial) node would represent one cluster within the partition of that particular layer. Using this structure we can propagate the information forward through the layers with the new artificial nodes and back again to the original graph.

In the forward propagation, we pass along the information of a node c at layer i to the node representing the cluster in which n is contained in layer $i + 1$. In

the backward propagation, we use the same connections, just in the opposite direction. Figure 6.8 displays the forward propagation through the artificial nodes, the backward propagation is left out in the Figure for simplicity.

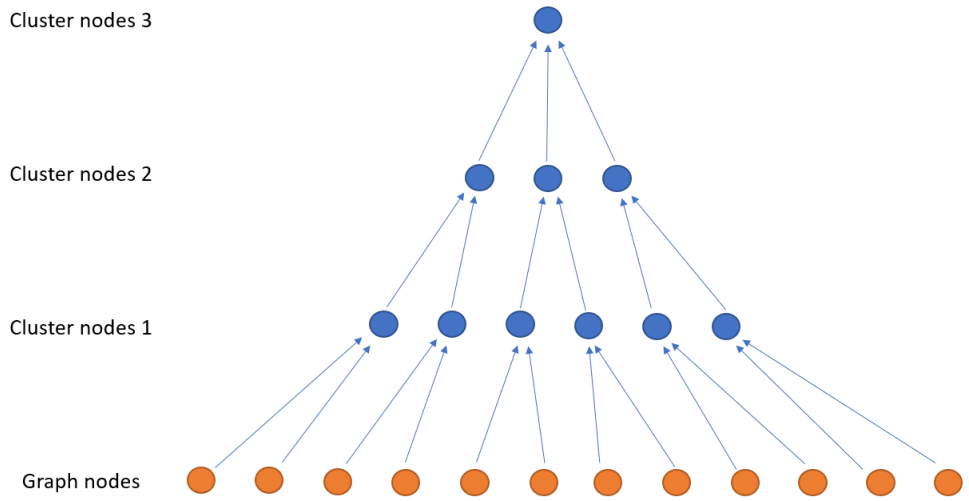

Figure 6.8: Brief overview of the forward propagation through the clusters. The orange nodes represent the original graph nodes whereas the blue nodes are the newly added nodes that represent the cluster.

To implement this idea we need to create partitions of the graph where each cluster in the partition would be represented as an artificial node. A popular method to create such partitions is the Louvain method, which extracts communities from large networks [23].

For the first layer, we apply the Louvain method on the original graph to obtain a partition of the nodes. We fine-tune the parameters of the Louvain method such that each cluster in the partition would contain between 3 to 8 nodes of the original graph. For the second layer, we construct a new graph containing only the partitions from the step before as nodes. The edges of this graph G are defined as $(u, v) \in G \iff (\exists (i, j) \in G_{original} \mid i \in u \ \ \& \ \ j \in v)$. We again apply the Louvain method to this newly constructed graph to obtain the second layer of the clustering.

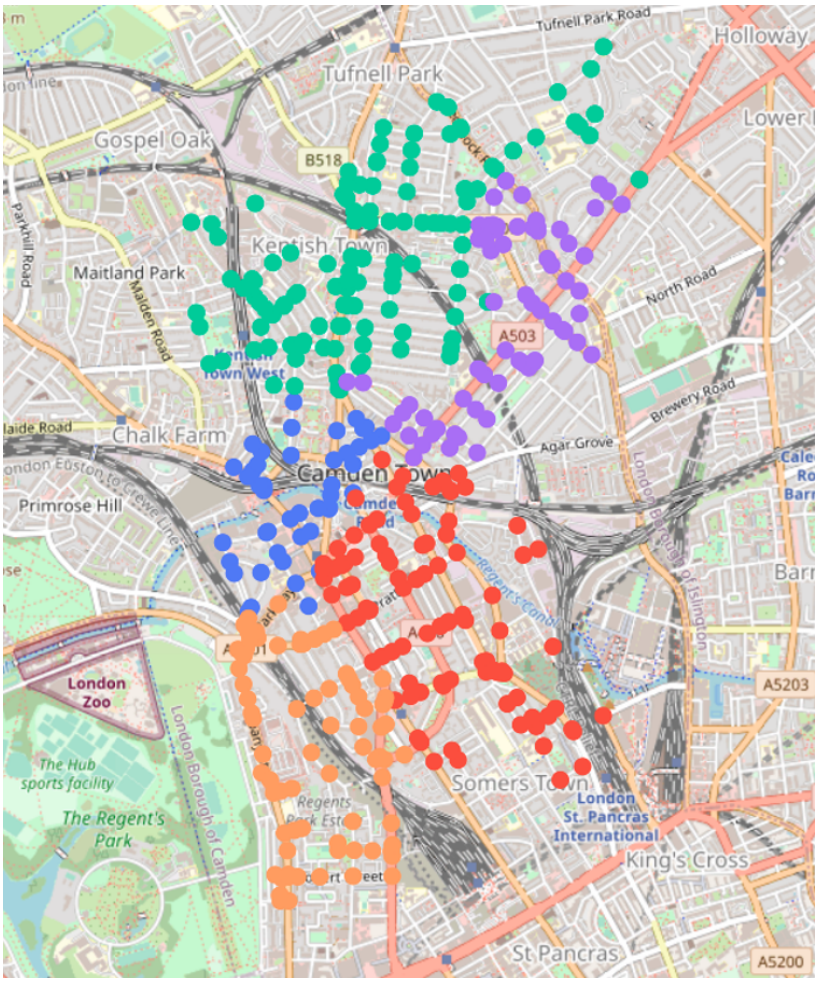

Figure 6.9: Five different clusters at layer 3 and their contained original graph nodes at the lowest level of the hierarchical structuring. Each of the colors in the Figure corresponds to one of the clusters.

We further repeat this procedure until we end up with the number of clusters we intend to use. Figure 6.9 shows the original graph nodes contained in 5 clusters of the partition at layer 3 as an example. In the first approach of the clustering, we use 3 layers for the propagation and later change that to 5. We include the cluster nodes in the GNN model by propagating the data up through all the cluster layers and back again after every GNN layer as written before. For these additional convolutions, we use the same setup like for the supersegment convolutions as we again have no edge attributes for the new edges.

Results

To test the new hierarchical structure with the clusters we define three runs. As the baseline, we have the original model described in 5 with the first approach of the supersegments. Secondly, we implement two different models of the clusters - One with 3 layers of clustering and one with 5 layers.

The model with the additional layers takes significantly more time to train. Therefore, we only have the results for after 38 epochs as we stop training the model after 48 hours of run time. We can see, that the clusters did not improve the performance at all and that the performance gets even worse when adding more layers.

Run	Epoch	Training loss
Baseline	38	0.8699
With 3 layer clustering	38	0.8730
With 5 layer clustering	38	0.8748

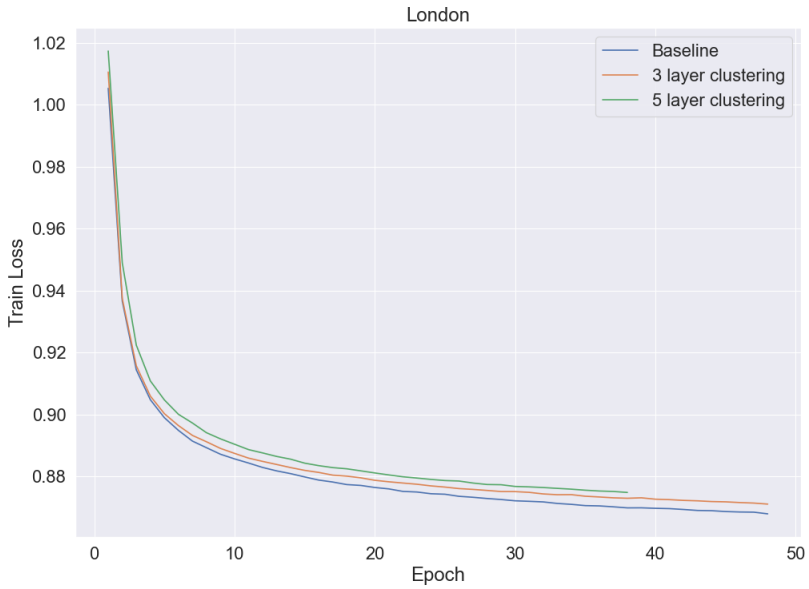

Figure 6.10: Training scores for this approach

6.3 Architecture tuning

6.3.1 Learning rate

Approach

Other submissions to the Traffic4Cast challenge achieved a similar performance with higher learning rates. As the training time of a single run is approximately two days we want to investigate if a higher learning rate can shorten the training time while maintaining the same performance. The original value for the learning rate in the baseline solution is $5e^{-4}$. We decide to test a greater and a smaller value for it to explore in both directions. For this experiment, we train the network apart from the baseline run with learning rates of $5e^{-5}$ and $1e^{-3}$.

Results

We test three different learning rates to figure out which one would work the best for our model. As the baseline, we define the original model with the learning rate set at $5e^{-4}$. We can see in Figure 6.11 that this is indeed the learning rate with which the performance is the best. With the higher learning rate, the model is not able to learn the correct prediction whereas with the lower learning rate the training time increases significantly and the results are still worse after 50 epochs. Note, that here the scores for Melbourne are displayed.

Run	Epoch	Training loss
Baseline	50	0.8905
$5e^{-5}$ as learning rate	50	0.8946
$1e^{-3}$ as learning rate	50	0.9419

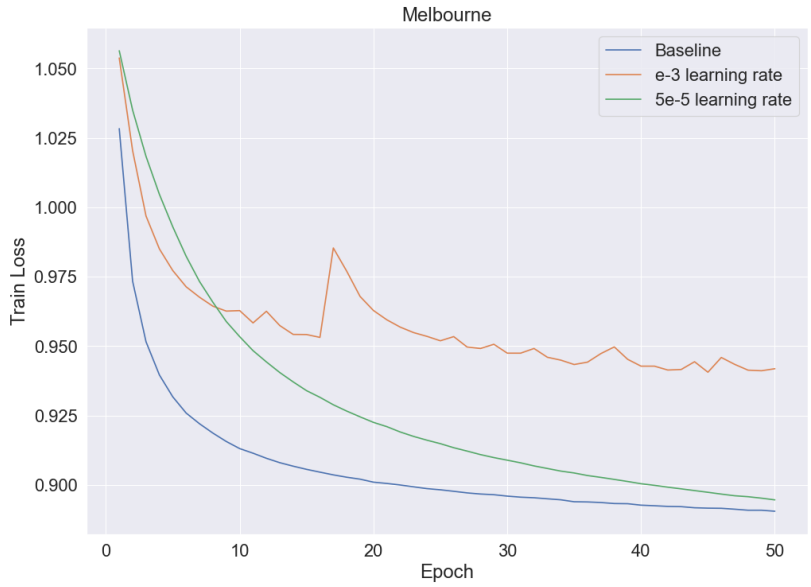

Figure 6.11: Training scores for this approach

6.3.2 GNN layers

Approach

With all the additional features mentioned before the required memory for the GNN to train increases dramatically. There is a clear trade-off between how far information between adjacent nodes can be exchanged and how much memory and training time the model uses. That is one of the main reasons for the parameter tuning on the GNN layers. We want to find out what the optimal compromise between computational expense and accuracy is. For that, we train the network with numbers of layers between 1 and 6 for comparison.

Results

To find out how well the network works with different numbers of GNN layers we use the baseline model with the supersegments approach 1 and train it with different numbers of GNN layers. We can see that the difference is small between four, five, and six layers, however, the training loss increases significantly when using less than four layers. One reason for this behavior could be the fact that the last two layers use a different convolution which does not use the edge features [3].

Run	Epoch	Training loss
6 GNN layers	50	0.8691
5 GNN layers	50	0.8697
4 GNN layers	50	0.8708
3 GNN layers	50	0.8753
2 GNN layers	50	0.8790
1 GNN layers	50	0.8851

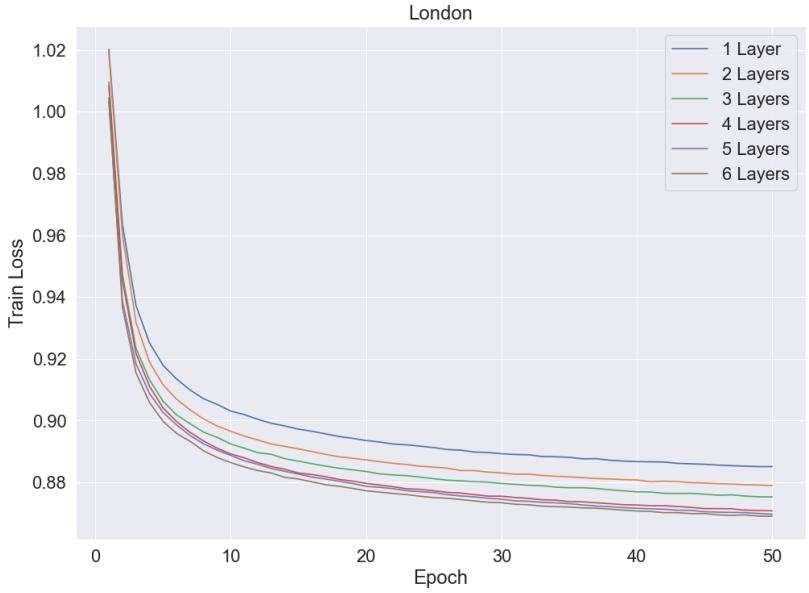

Figure 6.12: Training scores for this approach

6.3.3 GNN embedding size

Approach

For the same reasons as for the GNN layers and in order to find learning bottlenecks, we experiment with the embedding size of the GNN. We expect runs with a higher embedding size to use more computational power but perform better. Therefore we train the network with embedding sizes of 192, 256, and 384 for comparison.

Results

We use the baseline model with supersegments approach 1 for the comparison of different embedding sizes. For that, we train the model with embedding sizes

192, 256, and 384, where 256 is already the configuration of the baseline.

The training loss decreases significantly when using a larger embedding and shows us the greatest effect over all the approaches.

Run	Epoch	Training loss
embedding size 192	50	0.8751
embedding size 256	50	0.8700
embedding size 384	50	0.8602

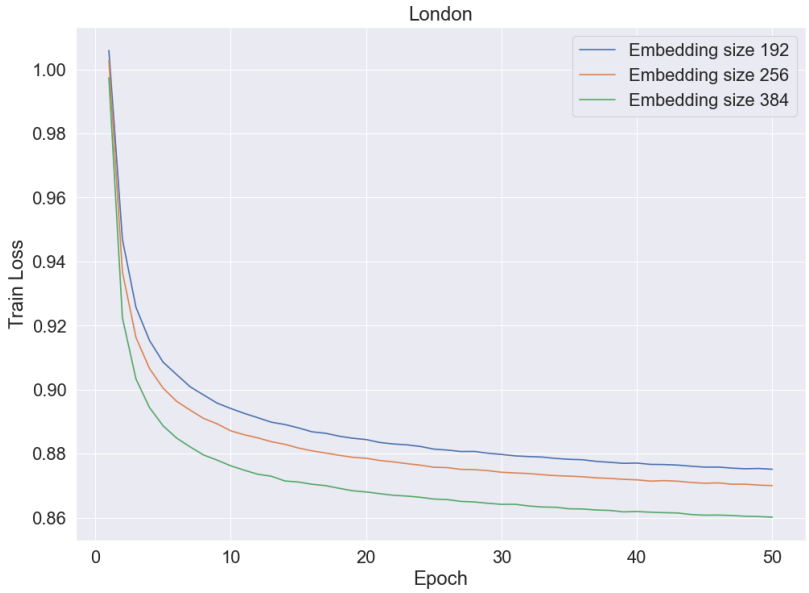

Figure 6.13: Training scores for this approach

6.4 Final configurations

Approach

With all the insights gained throughout the experiments, we want to combine the best approaches to create a final model.

Results

With the results shown before we combine all of the best approaches to create a final run for each city. We build that run again on top of the baseline model defined in 5 with their supersegments approach 1.

We include as additional node features the 4 PCA values and the airport distance for each node in the graph. We use the original supersegments without the clustering approach. For the parameters of the GNN, we change the embedding size of the GNN to 384, where the baseline uses only 256.

We can see, that for all cities there is a significant decrease in both training and validation loss. This is a result of all the changes made to the model. However, we can see, that for Melbourne the model seems to overfit as the validation loss is a lot higher than the training loss. Nevertheless, this seems to have been already the case with the baseline solution, and the validation loss decreases with the changes as well.

Run	Epoch	Training loss	Validation loss
London Baseline	50	0.8703	0.8778
London Best Configs	50	0.8496	0.8537
Madrid Baseline	46	0.8590	0.8591
Madrid Best Configs	46	0.8472	0.8511
Melbourne Baseline	50	0.8905	0.9130
Melbourne Best Configs	50	0.8665	0.8969

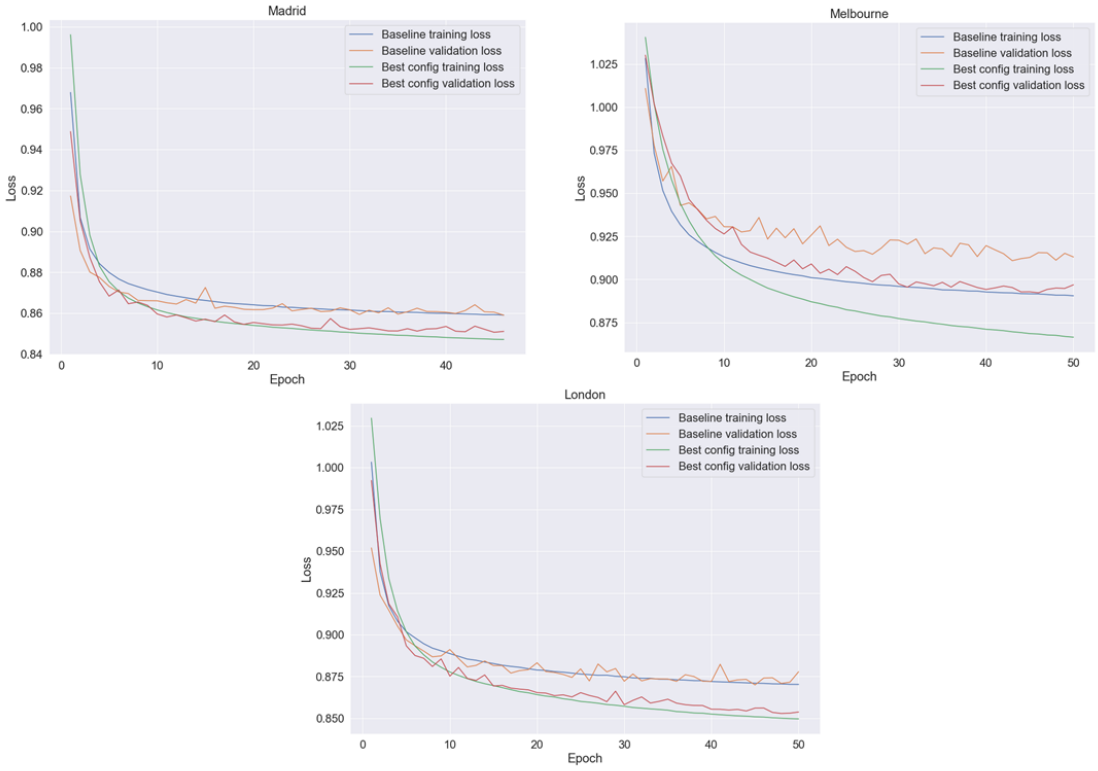

Figure 6.14: Training and Validation losses for the 3 cities. Comparing the baseline solution runs with a run that uses the best-found configurations.

Chapter 7

Conclusion

We used different approaches to experiment with the performance of the GNN predicting the congestion classes. From the experiments conducted, we can analyze the different approaches and draw conclusions.

The results show us, that with the new features added to the network the performance increases. In the experimental evaluation, only the training loss is included, however, the loss on the validation set looks similar for most of the runs. This shows, that the selected features are meaningful and can help to improve the model.

The experiments on the structure of the graph however did not improve the baseline model. We see in the case of the supersegments, that the choice of the nodes for them does not have a large impact on the performance of the network. Adding a hierarchical structure on top of the original road graph did not bring the expected results either. This shows that creating more connections and spreading information more globally does not always help the performance.

In the architecture tuning, we see with the experiments, that often the mere use of more computational power and memory leads to better results. The largest increase in performance, in the end, came from using a larger embedding size for the GNN.

Our goal is to improve a baseline solution [3] to the Traffic4Cast challenge by applying multiple different approaches. Looking at the total results of our experiments we can see, that we have indeed improved the solution. However, one has to note that a large part of this improvement came from sheer computational power by increasing the embedding size of the GNN.

For future work, it would be interesting to investigate further how the structure of the graph influences the quality of the predictions. One of the other submissions to the Traffic4Cast challenge is using multiple models to then take the average of the prediction [16] and report great results with this approach.

This is something that would be interesting to experiment with, especially combined with different graph structures.

We trained a separate model for each city to make the predictions. Another interesting approach that we could not experiment with due to memory limits is to train one model for all the cities combined.

Bibliography

[1] Editor, "Traffic prediction: How machine learning helps forecast congestions and plan optimal routes," Jan 2022. [Online]. Available: https://www.altexsoft.com/blog/traffic-prediction/

[2] Iarai, "Iarai/neurips2022-traffic4cast." [Online]. Available: https://github.com/iarai/NeurIPS2022-traffic4cast

[3] F. Grötschla and J. Mathys, "Hierarchical graph structures for congestion and eta prediction," 2022. [Online]. Available: https://arxiv.org/abs/2211.11762

[4] E. Vlahogianni, M. Karlaftis, and J. Golias, "Short-term traffic forecasting: Where we are and where we're going," *Transportation Research Part C: Emerging Technologies*, vol. 43, 06 2014.

[5] A. Daigavane, B. Ravindran, and G. Aggarwal, "Understanding convolutions on graphs," *Distill*, 2021, https://distill.pub/2021/understanding-gnns.

[6] [Online]. Available: https://www.openstreetmap.org/about

[7] "Traffic4cast." [Online]. Available: https://www.iarai.ac.at/traffic4cast/

[8] "Chapter 13 - graph neural networks." [Online]. Available: https://disco.ethz.ch/courses/fs22/podc/lecturenotes/chapter13.pdf

[9] A. Dubey, "The mathematics behind principal component analysis," Sep 2022. [Online]. Available: https://towardsdatascience.com/the-mathematics-behind-principal-component-analysis-fff2d7f4b643

[10] "A step-by-step explanation of principal component analysis (pca)." [Online]. Available: https://builtin.com/data-science/step-step-explanation-principal-component-analysis

[11] E. I. Vlahogianni, M. G. Karlaftis, and J. C. Golias, "Short-term traffic forecasting: Where we are and where we're going," *Transportation Research Part C: Emerging Technologies*, vol. 43, pp. 3–19, 2014, special Issue on Short-term Traffic Flow Forecasting. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X14000096

[12] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Graph convolutional recurrent neural network: Data-driven traffic forecasting," 07 2017.

[13] A. Ermagun and D. Levinson, "Spatiotemporal traffic forecasting: review and proposed directions," *Transport Reviews*, vol. 38, no. 6, pp. 786–814, 2018. [Online]. Available: https://doi.org/10.1080/01441647.2018.1442887

[14] D. Pavlyuk, "Feature selection and extraction in spatiotemporal traffic forecasting: A systematic literature review - european transport research review," Jan 2019. [Online]. Available: https://link.springer.com/article/10.1186/s12544-019-0345-9#citeas

[15] W. Zhang, Y. Yu, Y. Qi, F. Shu, and Y. Wang, "Short-term traffic flow prediction based on spatio-temporal analysis and cnn deep learning," *Transportmetrica A: Transport Science*, vol. 15, no. 2, pp. 1688–1711, 2019. [Online]. Available: https://doi.org/10.1080/23249935.2019.1637966

[16] L. Deng, C. Wu, D. Lian, and M. Zhou, "Transposed variational auto-encoder with intrinsic feature learning for traffic forecasting," 2022. [Online]. Available: https://arxiv.org/abs/2211.00641

[17] M. Lumiste and A. Ilie, "Large scale traffic forecasting with gradient boosting, traffic4cast 2022 challenge," 2022. [Online]. Available: https://arxiv.org/abs/2211.00157

[18] Z. Zhong, C.-T. Li, and J. Pang, "Hierarchical message-passing graph neural networks - data mining and knowledge discovery," Nov 2022. [Online]. Available: https://link.springer.com/article/10.1007/s10618-022-00890-9

[19] L. Yang and W. Huang, "Representation and assessment of spatial design using a hierarchical graph neural network: Classification of shopping center types," *Automation in Construction*, vol. 147, p. 104727, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0926580522005970

[20] A. Partow, "The global airport database." [Online]. Available: https://www.partow.net/miscellaneous/airportdatabase/

[21] //facebook.com/vitalflux, "Pca explained variance concepts with python example," Aug 2022. [Online]. Available: https://vitalflux.com/pca-explained-variance-concept-python-example/#:~:text=Conclusion-,WhatisExplainedVariance,componentanalysis(PCA)method.

[22] "Iarai/neurips2022-traffic4cast." [Online]. Available: https://github.com/iarai/NeurIPS2022-traffic4cast/blob/main/blogposts/20220905_More_on_supersegments_and_ETAs.md

[23] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical*

Mechanics: Theory and Experiment, vol. 2008, no. 10, p. P10008, oct 2008. [Online]. Available: https://dx.doi.org/10.1088/1742-5468/2008/10/P10008

Appendix A

Dataset statistics

We look into the given dataset in more detail to address potential issues or challenges when working with this dataset. The Traffic4Cast challenge provides us with data for 3 cities - London, Madrid, and Melbourne. For each of these cities, we have the complete road graph as a static file from which we build our graph.

A.1 Sparsity

As an input to predict the congestion classes for all OpenStreetMap edges in these cities, we are getting spatially sparse car counter data. To be precise, for each point in time, we are getting 4 values that represent the car counter values for the last 1 hour in 15-minute intervals. The data is spatially sparse as for example in London only around 6% of the nodes have a counter and even nodes with a counter do not always have values for a specific point in time.

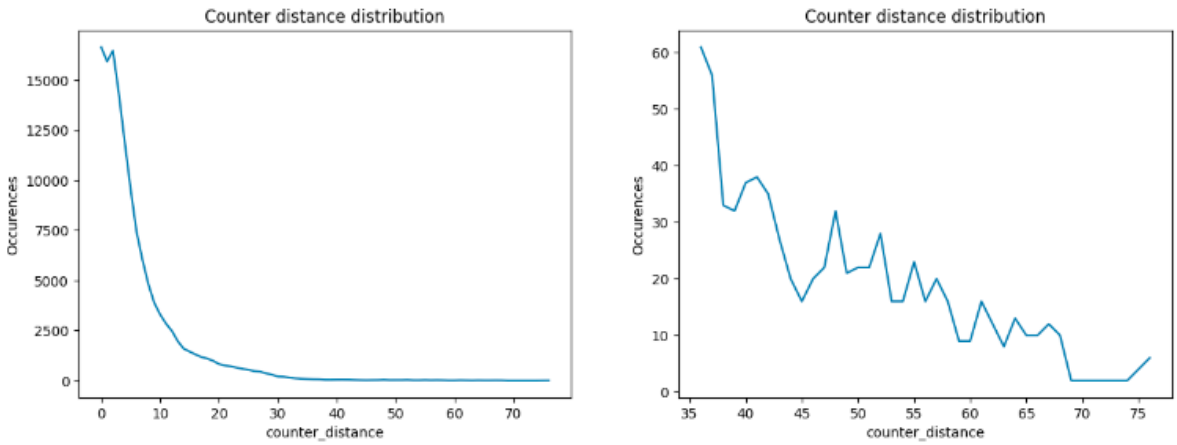

Figure A.1: The distribution of the distance to the next counter node.

Another statistic that describes well the sparsity of the input data can be seen in Figure A.1. We see that although most of the nodes in the graph are close to a counter there is still a significant amount of nodes that have a counter distance greater than 35. It takes a lot of iterations in the message-passing process until they get reached what might be problematic.

One part of the given data is the ground truth labels corresponding to the given input data. The labels are always in the set {1, 2, 3}, indicating the traffic congestion on an edge at a given moment with 3 being highly congested and 1 no congestion at all. Here again, sparsity is a problem as most of the edges only have few ground truth labels to them. An edge can have up to 96 ground truth labels per day as we split the time into 15-minute intervals.

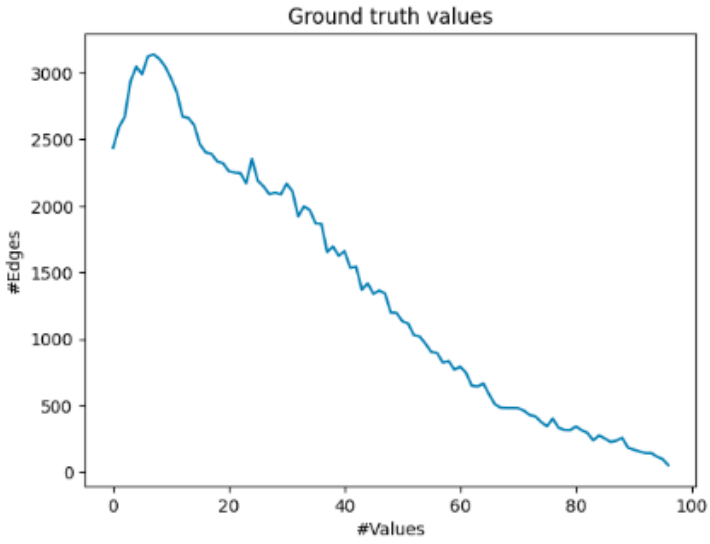

Figure A.2: Distribution of the number of ground truth values over the complete road graph in London on the 2020-01-30.

In Figure A.2 we can see the distribution of the ground truth values over the edges in London for the time of one day. We see, that the peak is around 10 ground truth values per edge per day. Also, it is worth noting, that 18% of all the edges in the graph did not have any ground truth values on that specific day.

A.2 Edge lengths

Another thing to consider in the input data is the length of the edges in the graph. OpenStreetMaps represents curves as multiple short edges which results in a lot of edges with a length even smaller than a car.

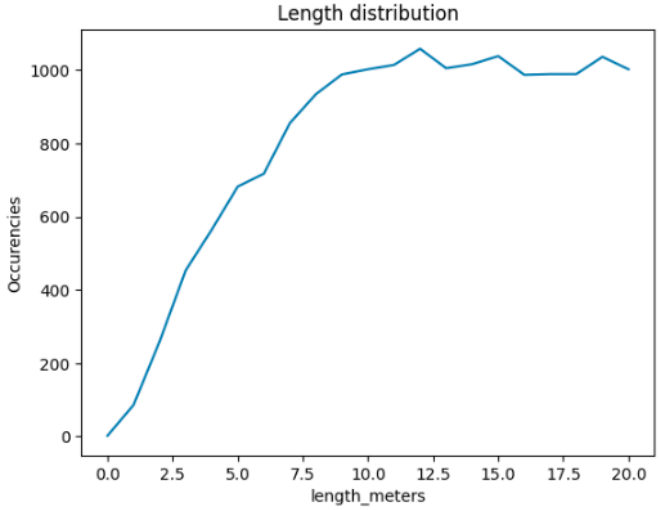

Figure A.3: Distribution of the length of the edges in meters. Note, that only the range from 0 to 10 meters is shown here.

We can see this fact in Figure A.3. Making predictions for edges with a length of less than 5 meters can be inaccurate. One option to tackle this is to concatenate the short edges, however, one must be careful to still include the intermediate nodes in the computation of the predictions.

Appendix B

Setup

The runs are made on two different types of GPUs. Most of the simpler, less expensive runs are made on a GeForce RTX 3090 where as the later runs are made on an RTX A6000. The training time depends on the complexity of the added features. We use a maximum training time of 48 hours, which already was not enough for some of the experiments. For most of the runs we used 24GB of memory, whereas, for the more expensive experiments, the usage was at 40GB. One of the largest limitations was indeed the memory as the models ran out of it pretty quickly when dealing with a lot of data and features.

Appendix C

Final configurations

For the final configurations that we use in 6.4, we have the following setup:

- We are using 19 node features in total: 8 global features, 4 traffic counter values, 2 features for the GPS coordinates, 4 features for the principal components of the PCA, and one feature for the normalized distance to the next airport.
- We are using approach 1 for the supersegments as described in 5.1.2 with the original supersegments.
- We do not use any other modifications on the graph structure.
- We are using an embedding size of 384 in the GNN.
- We are using 6 GNN layers.