

# **Predicting Traffic Flow Using Deep Learning**

Oliver Wiech

MSc Machine Learning in Science

University of Nottingham

August 20, 2021

## ABSTRACT

This paper presents and evaluates deep learning architectures for traffic flow prediction task based on Traffic4Cast Challenge data published by the Institute of Advanced Research in Artificial Intelligence. Traffic density consists of both spatial and temporal correlations throughout the road network that is represented in the form of an image, a matrix or a graph. In order to predict the next one hour of timesteps of the traffic volumes and speeds Convolutional Neural Network (CNN) and a Graph Convolutional Neural Network (GCNN) are suggested, evaluated and compared putting the emphasis on different types of convolutional layers: Chebyshev, Hypergraph and Generalised Graph convolution. The models are trained on data of four major cities and have their generalisation capabilities tested on two unseen areas, as well as the results are submitted to the Traffic4Cast Challenge. The study concludes that the GCNN models are better in the traffic flow prediction task in terms of both the performance and the efficiency than the U-Net CNNs. The GCNN results indicate that it could be reliably used in real-life Intelligent Transportation Systems in order to reduce congestion and help individuals and businesses to manage the traffic more efficiently.

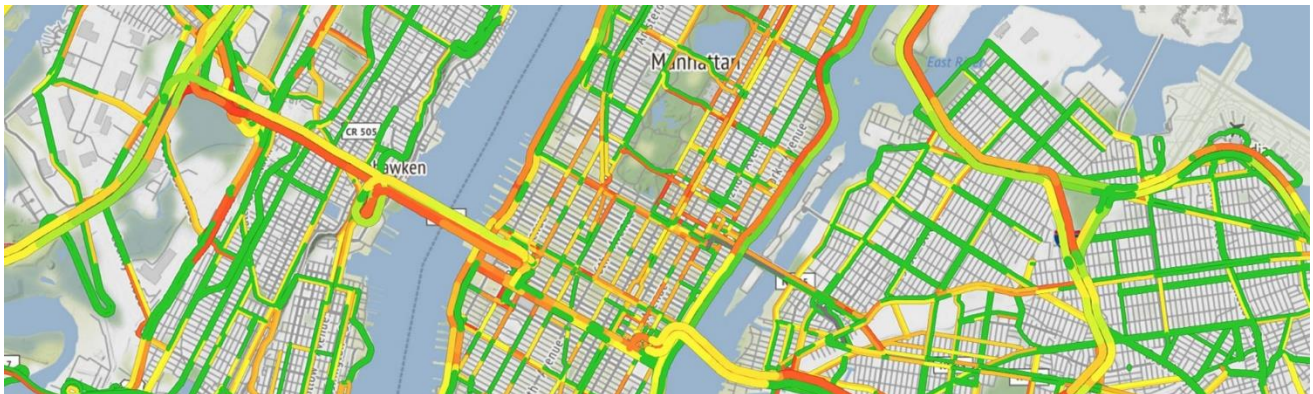
## Table of Contents

<b>ABSTRACT .....</b>	<b>2</b>
<b>TABLE OF CONTENTS .....</b>	<b>3</b>
<b>CHAPTER I: INTRODUCTION .....</b>	<b>4</b>
1.1 Intelligent Transportation Systems and Traffic4Cast Challenge .....	4
1.2 The Machine Learning of Traffic Flow Prediction .....	5
<b>CHAPTER II: METHODS AND METHODOLOGY.....</b>	<b>6</b>
2.1 Dataset and Image Augmentation .....	6
2.1.1 Core Challenge Overview.....	6
2.1.2 Dynamic Data .....	7
2.1.3 Static and Graphs Data.....	9
2.1.4 Data Preprocessing and Technical Issues .....	10
2.2 Methods and Architectures .....	13
2.2.1 Convolutional U-Net.....	13
2.2.2 Graph Convolutional Neural Network.....	16
2.2.3 Naïve and Weighted Averages.....	20
2.2.4 Optimizer and Hyperparameters .....	21
<b>CHAPTER III: RESULTS AND DISCUSSION.....</b>	<b>24</b>
3.1 Results.....	24
3.2 Discussion.....	31
<b>CHAPTER IV: CONCLUSION .....</b>	<b>34</b>
<b>REFERENCES.....</b>	<b>36</b>

# CHAPTER I: INTRODUCTION

## 1.1 Intelligent Transportation Systems and Traffic4Cast Challenge

Road congestion is a serious challenge in the modern world where an increasing number of people own a vehicle. It leads to traffic delays, which impact the economy and fuel environmental damage. Systems observing traffic trends and predicting traffic density such as intelligent transportation systems (ITS) address these issues. ITS aims to provide reliable and innovative services related to transport and traffic management. The users, that are individuals, businesses or governments, are better informed about the road network and its traffic flow density, allowing them to make better and more coordinated decisions. Solutions like these have the potential of reducing traffic congestion, leading to a better quality of life and planet health due to the reduction of the traffic congestion influence on global warming [1], as well as reducing the costs for public traffic companies by helping them optimize the traffic routes depending on the predicted congestion as shown on Fig. 1.



*Figure 1 Reliable traffic flow prediction can help in public transport management and handling traffic bottlenecks in congested cities like New York [2].*

To help with the development of efficient traffic flow prediction solutions, the Institute of Advanced Research in Artificial Intelligence together with mapping and location data company HERE Technologies has developed Traffic4Cast Challenge [3]. In which the researchers from all around the world are given a common training and testing framework to compare the results of different machine learning approaches for predicting the vehicles' traffic flow. The Challenge had editions in 2019, where the winner was S. Choi with his U-Net approach, and in 2020 where H. Martin won by using the modified Graph ResNet architecture. Those two models are the basis of our current work, where we expand them and try them out in different

configurations to improve the results in this year edition of Traffic4Cast which focuses on robust shifting in space and time of the data.

## 1.2 The Machine Learning of Traffic Flow Prediction

Machine learning techniques have been gaining traction over the years. Neural networks, consisting of neurons resembling ones found in the human brain are capable of predicting outputs based on features they were trained with [4]. They have developed from statistical methods, also used for traffic prediction, like support vector machines (SVM). Spatial and temporal features present in road traffic require the use of convolutional neural networks (CNN) which are able to capture them [5]. Normally, representing the traffic in a time series format to capture the temporal dynamics requires the use of long-short term memory (LSTM) recurrent neural networks (RNN) which are not needed in our approach due to collapsing the time and channels dimensions together allowing for CNNs and GraphCNNs to handle the temporal changes without the need of RNN modules.

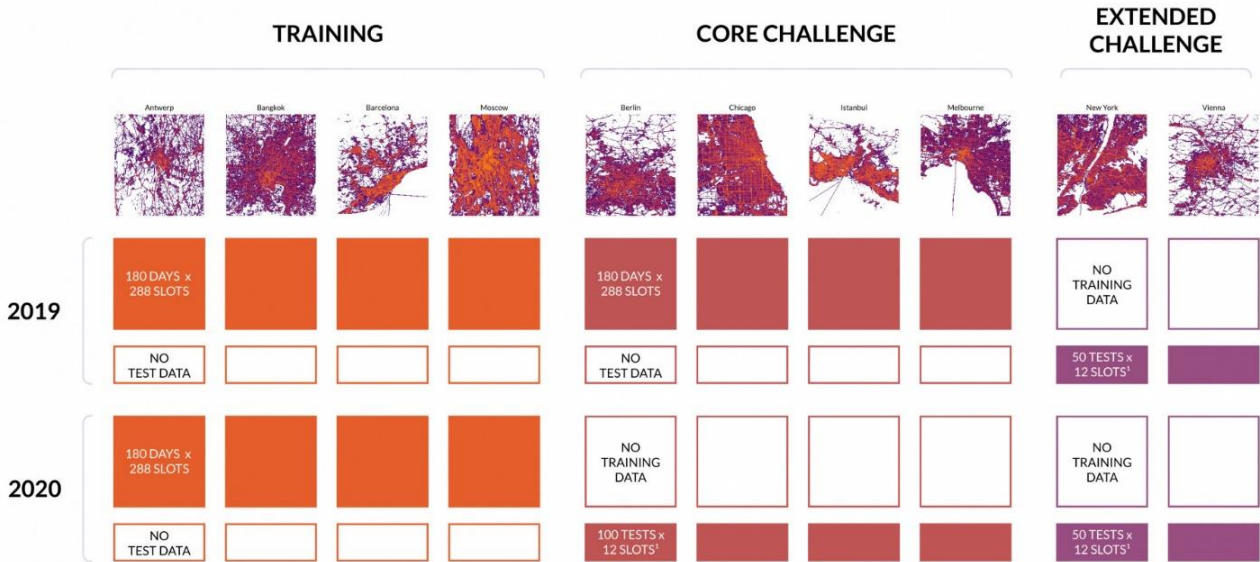
The input to a CNN and a Graph CNN is very different. In the former, it is a matrix of pixels that represent an image containing the information about the volume and speed of the traffic flow, while the latter uses a graph structure of the road network which gives the information about the connectivity of the road and enhances the efficiency of the training by reducing the number of pixel information needed – in the CNN approach often the majority of the image pixels are equal to zero since there is no roads, or traffic flow at that particular time, on that specific area.

This paper investigates the use of machine learning to predict road congestion across different major cities in the world focusing on the two deep learning architectures: convolutional neural networks and graph convolutional neural networks. A real traffic dataset is used that is published by the Institute of Advanced Research in Artificial Intelligence. The performance of neural networks is evaluated and summarised. Spatial and temporal relationships are compared, with the conclusions drawn on the models' performance and thorough discussion unfolded.

## CHAPTER II: METHODS AND METHODOLOGY

### 2.1 Dataset and Image Augmentation

The dataset given by IARAI, and summarized in Figure 2, consists of data for ten different cities with a split in the dataset between core and extended challenges. The training samples were gathered using a large number of probe vehicles equipped with GPS modules with a time interval being equal to 5 minutes. This project is focused mostly on the core challenge with the data outside of the core challenge being used in analyzing the models' performances with the emphasis on the abilities to generalize into new unseen before city road network.



<sup>1</sup>Given one hour of data in 12 slots of 5min, predict the next 5, 10, 15, 30, 45 and 60 lead times.

Figure 2 The overview of the data in the Traffic4Cast Challenge. The project is mostly concerned with the Core Challenge cities: Berlin, Chicago, Istanbul and Melbourne, however submissions are made for Extended Challenge as well [6].

#### 2.1.1 Core Challenge Overview

The core challenge uses the data from Berlin, Chicago, Istanbul, and Melbourne cities. While the additional training set has data from both 2019 and 2020, the core challenge cities are given only pre-Covid data for training purposes and the testing being done using post-Covid data, making this challenge focused on the traffic temporal domain shift due to the pandemic. The

data for each of the four cities consist of 180 data files, each equal to one day in 2019 starting from January 1<sup>st</sup> to June 30<sup>th</sup>. Furthermore, each day is split into 288 time slots with 5-minute intervals between every frame. The training data described here is called dynamic data due to its constant change over time. Additionally, a static data file is provided to create a graph representation of the road network and its traffic flow.

### 2.1.2 Dynamic Data

The dynamic data, as mentioned above, consists of 180 days with 288 time slots making it equal to 51 840 samples per each city in the Core Challenge set. Each day file is a large eight-dimensional tensor with the following shape: [288, 495, 436, 8], where 288 is the number of samples in a day, 495 and 436 being the height and width of the image, and 8 being equal to the number of channels per image. The channels provide information about speed and volume for the North-East, South-East, South-West and North-West directions, as shown in Figure 3.



*Figure 3 The dynamic data represents a timestep by a tensor of [width, height, channels]. The images above show the channels which are: North-West, North-East, South-West and South-East speeds and volumes respectively.*

Both the speed and volumes are normalized to take values between 0 and 255. Since the speeds usually have very low values, they are barely visible on the images due to having low pixel

intensity. Figure 4 visualizes typical values of the average of speeds and volume over the entire day traffic flow at a pixel point in the city centre of Berlin. We can see how the volume (blue bars) spikes up during rush hours – when people go to and from work, and the volume being low during late night and early morning time. The speed (red line) is consistent throughout the day but spikes up and down throughout the night due to lack of traffic flow in that time at this specific pixel point.

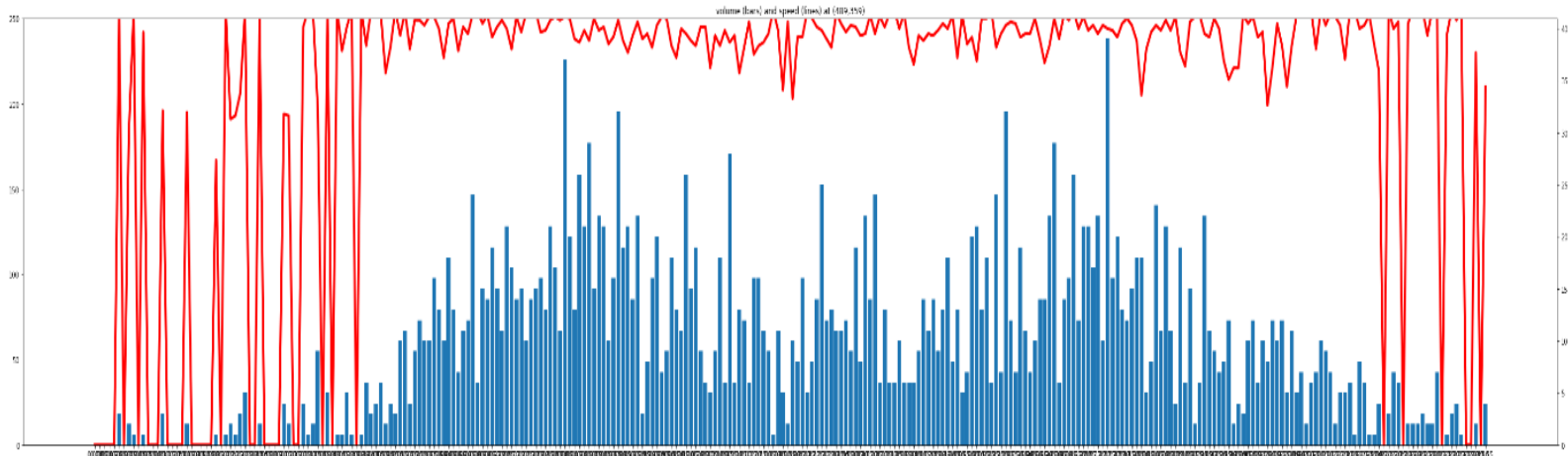


Figure 4 The volume (blue bars) and speed (red line) for a pixel point of (419, 159) coordinates in Berlin city calculated for the entire day. The speeds go to zero when the volume is also none. The volume spikes up during the rush hours and is very low during the night.

Also, an interesting observation found in the additional training data provided is the difference in volumes between pre-Covid and post-Covid time. Figure 5 presents the average pixel volume over all the timeslots for a day in 2019 and a day in the 2020 year. The post-Covid day has a relatively big decrease in volume compared to the previous year.

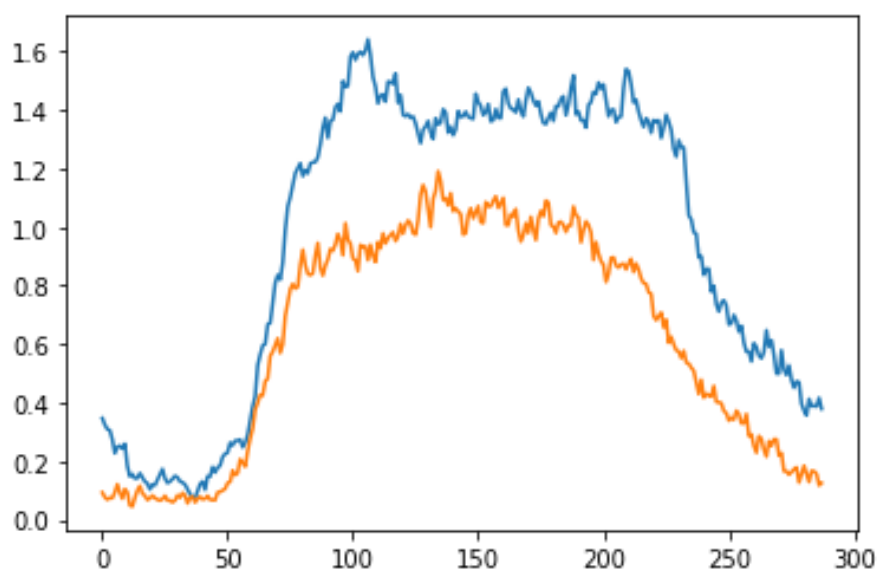
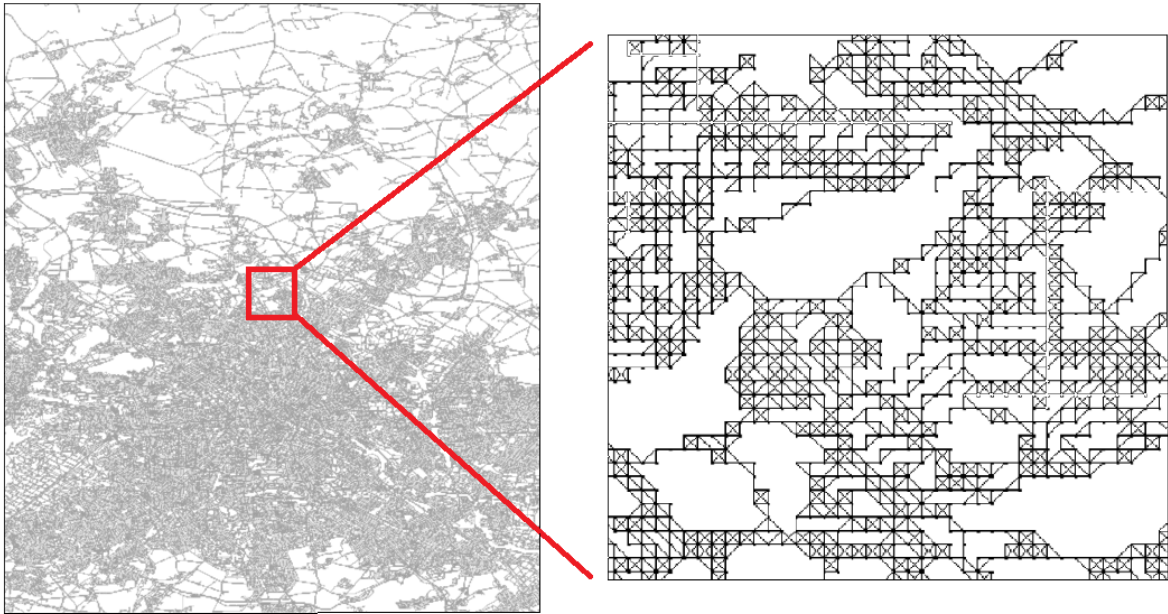


Figure 5 The additional data for Antwerp shows very different volumes for pre-covid (blue line) and post-covid (orange line) times. The difference can be explained by the lockdowns and work-from-home trend present in 2020.



### 2.1.3 Static and Graph Data

The static data contains a (9, 495, 436) tensor. The first channel contains information on the grey-scale representation of the city with the same resolution as the dynamic data. This channel and the next 8 channels were created by rasterizing a large 4950x4360 pixel resolution map of each city. Each consecutive channel is a down-sampled, lower resolution raster which is used to roughly represent the neighbourhood connectivity for every cell in the road network, as shown in Fig. 6.



*Figure 6 The neighbourhood connectivity is represented by large resolution image calculated from the static data different resolution rasters of the city map. The right image shows a zoomed-in part of the connectivity map.*

This static data is used to build a graph representation of the dynamic data that then is used in the form of an adjacency matrix as input to the Graphical Convolutional Network. The rasters use Moore neighbourhood in a pixel in order to introduce the edge on the graph. If there is a path with less than 7 neighbours active, an additional detour connection is created in order to properly catch the connectivity across corners in the Moore neighbourhood. This is a necessary addition in order to fix the issue where the GPS frequency on a point is low and only direct passes are caught, like in Fig. 7 where the orange points are not connected directly but due to a low frequency GPS points could move more directly between them, hence the green connection is added.

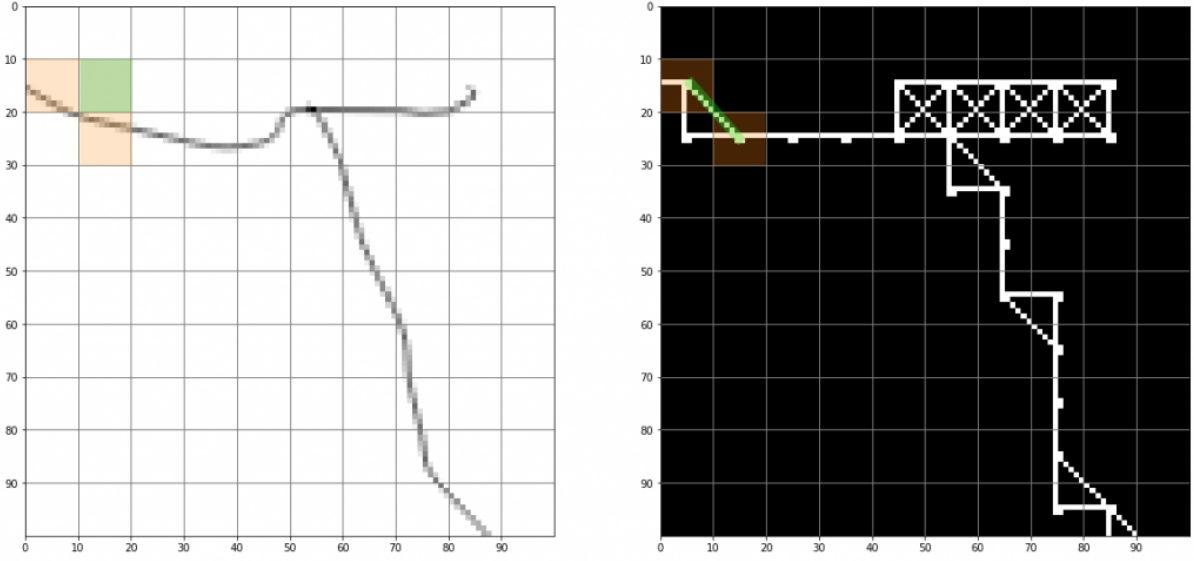


Figure 7 Adding additional connections (green line) Is crucial to properly represent the Moore neighbourhood connectivity across the corners. Depending on GPS frequency, the GPS point could move directly between the orange points [7].

#### 2.1.4 Data Preprocessing and Technical Issues

Dealing with such a large amount of data (a single timestep of dynamic data has  $495 * 436 * 8$  parameters, where there are 288 timesteps per day and 180 days per city which totals into 90 billion of pixel data for each city) requires a trade-off in terms of data preprocessing and the training done.

Therefore, to handle the resource issues (like Out Of Memory errors and very long training on full data) we had to reduce the data dimension for the U-Net parameters testing stage. The eight data channels that represent the speed and volume for the four directions were averaged to create only two channels instead of eight, as shown in Fig. 8, making the number of parameters four times smaller. Furthermore, the images were downsampled from the original resolution of 495 height, 436 width into 200 height and 200 width, making the optimal parameters search even quicker. The data was then normalized by division by 255, making the pixel value range from 0 to 1. This normalization procedure is often used, especially for sigmoid activations due to its positive effect on training – with larger values outside a small range of values where the sigmoid activations are strongest the derivatives can be small and resulted in slowed training. The lower values also help with the exploding gradient issue that can occur during the training process. The normalized and downsampled image can be seen in Fig. 9.

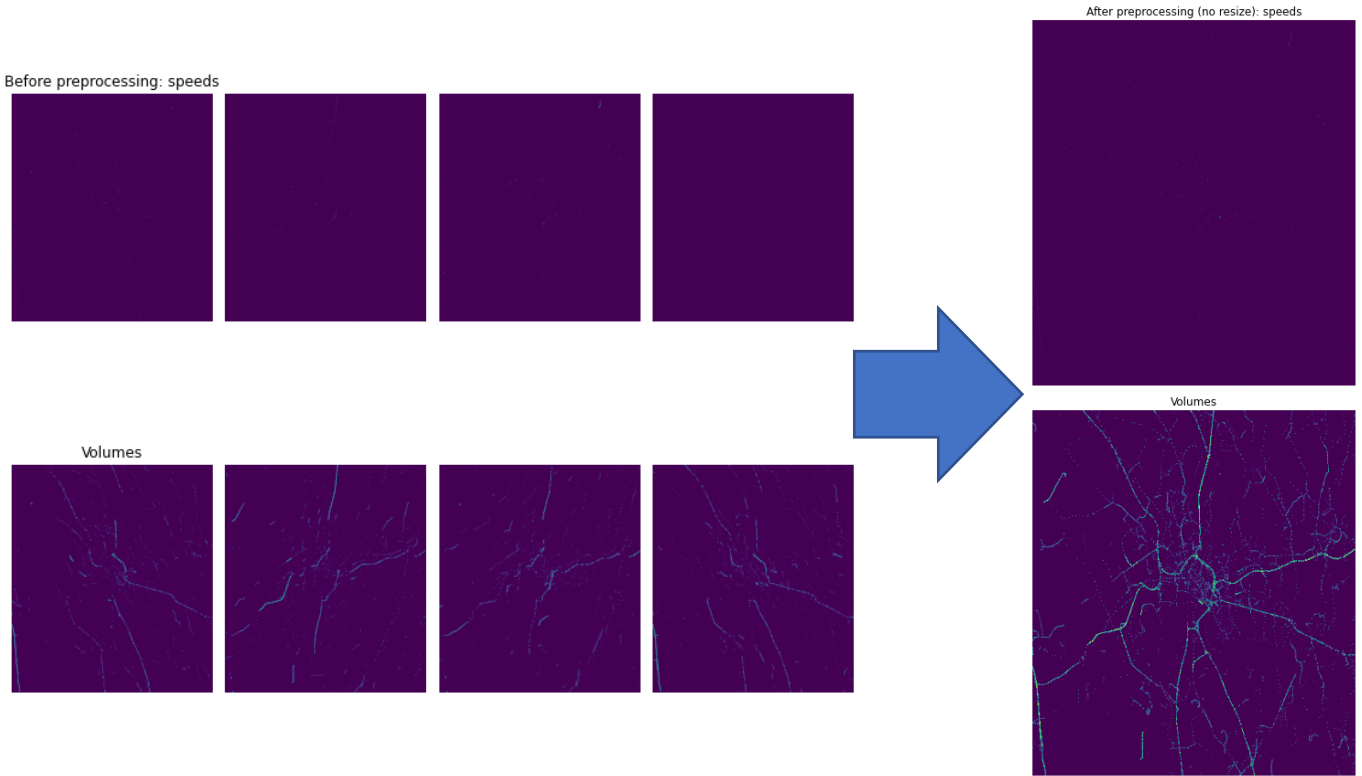


Figure 8 The eight speeds and volumes channels are averaged in order to make the training faster and reduce the out of memory issues.

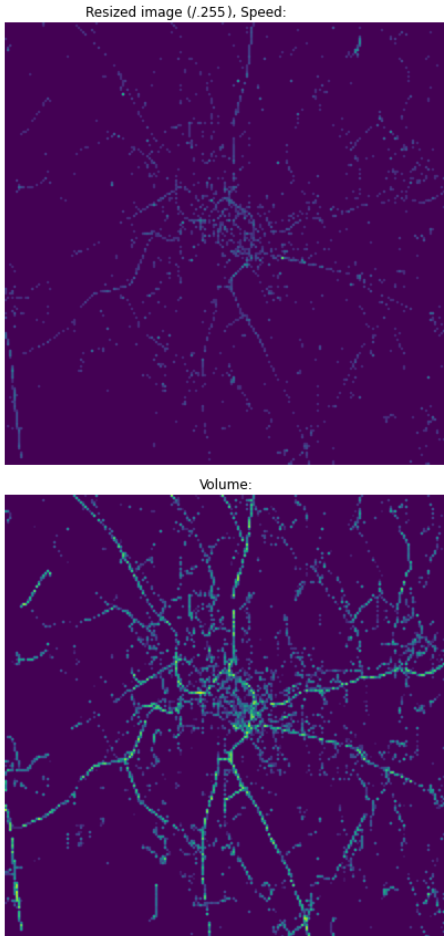


Figure 9 The original images are further downscaled to 200x200 resolution and normalised by 255 division.

The preprocessed data for experimenting cities, mostly Antwerp, was used for parameter optimization in U-Net. Despite all the steps, we encountered several Out Of Memory issues. To solve them and proceed with the task, we had to develop a custom Dataset Loader object based on Tensorflow’s Dataset class [8]. The custom data loader iteratively loads data files on a file-by-file basis during the training and sends the loaded .h5 files data into the preprocessing step. There is never more than one file in the memory which allows using even very large datasets efficiently and without problems.

One loaded file as described earlier is a tensor of (288, 495, 436, 8) where the first number is the number of 5-minute timesteps resulting in  $288 * 5$  minutes of data which is equal to exactly 24 hours. The data is split into the training data  $X$  and labels  $Y$  by taking a sequence of length equal to 12 timesteps (one hour) and then the next 6 timesteps (30 minutes) are saved as the response  $Y$  to that given sequence  $X$ . To even further decrease the number of training data without having a big impact on the model performance

we create a function to randomly take 18 sequences from each data file. The function creates a set of random 18 integers between 0 and  $(288*8)-(12*8+6*8)$  which then are used as the starting points of the 1-hour  $X$  sequences. The 288, 12 and 6 represent the number of 5-minute timesteps in an entire file,  $X$  sequence and its  $Y$  response respectively. They are multiplied by the number of channels (8) due to the introduction of collapsing time dimension into the channel dimension. This approach, which is used both – in the dynamic data for U-Net and also in the Graph Net data pre-processing step, allows us to use the U-Net and Graph Net without the specific need of recurrent blocks like LSTMs in order to capture the temporal dynamics of the traffic sequence since the time now is directly embedded into the U-Net/Graph Net input channels.

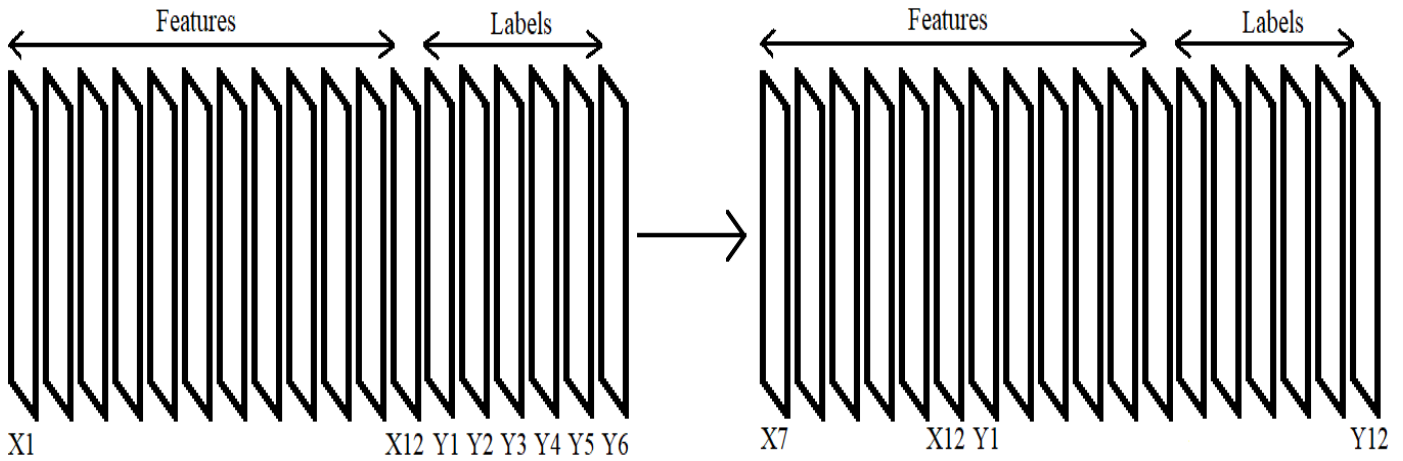


Figure 10 The original input sequence (X1-X12) is used to get the prediction output (Y1-Y6). To get the next half an hour of predictions, we stack the previous output (Y1-Y6) into the last half of the initial input (X7-X12).

The Traffic4Cast Challenge requires predictions for 5, 10, 15, 30, 45 and 60-minute predictions into the future after the end of the given test 1-hour sequence, while the output sequence of a model is only the next 6 5-minute time steps. Fig. 10 presents how we got 45 and 60-minute predictions for U-Net and Graph Net: after inputting the original test sequence (X1-X12) we take the prediction output (Y1-Y6) and add it to the end of the necessary amount of previous input to form a new one-hour input sequence. Y1-Y6 is added to the end of X7-X12 timesteps to form a 1-hour sequence that is then used as the new input to the model which outputs the 35, 40, 45, 50, 55 and 60-minute traffic flow predictions after the last point of the initial testing sequence. This allows us to pick the necessary for the challenge timesteps of predictions and form the final submission.

## 2.2 Methods and Architectures

In order to take part in the Traffic4Cast competition and solve the traffic prediction challenge we analysed and improved the performance of the two winning architectures of Traffic4Cast 2019 and 2020: a Convolutional Neural Network U-Net architecture, and a Graph Convolutional Neural Network architecture mostly based on Graph ResNet.

### 2.2.1 Convolutional U-Net

Originally U-Net was developed by the University of Freiburg in 2015 for biomedical image segmentation tasks [9]. It takes an image as an input and outputs another image, where in our case the input is a tensor of (495, 436, 96) where the 96 is equal to the sequence of length 12 multiplied by the 8 channels per timestep and the output is equal to a tensor of (495, 436, 48) with added small padding described later. The 48 channels of the output are equal to the sequence of traffic flow predicted for the 30 next minutes after the last input timestep.

The U-net consists of two parts as shown in Fig. 12: contracting path and an expansive path where they work similarly to a decoder and encoder constructions from Generative Adversarial Networks. The contracting path downsamples the input tensor by applying max-pooling after each convolution-dropout-convolution block. It allows the model to understand the crucial part of the input data representation. The downsampled tensor is then used as an input to the expansive path part, which works in the opposite way of the contracting path: it upsamples the input by using the transpose convolution layers. The number of feature channels which was greatly increased at each step in the contracting path now is getting reduced during the up-sampling process. The final layer is a convolution that outputs the desired image.

The contracting path typical block as mentioned earlier is convolution-dropout-convolution followed by max-pooling. The convolutional layer applies a kernel to the image matrix to transform the image and identify the features in the image matrix [10]. The number of kernels increases as we go deeper in the contracting path meaning the number of features and also the dimensions increases as well. Since we are mostly interested in the pixels that have the biggest impact on the activation, the max-pooling takes the maximum value over a given area and discarding the rest values. The dropout between the convolutions randomly deactivates the neurons to increase the generalisation capabilities and counter the overfitting issue [11]. In summary, the contracting block uses convolutions to get as many features as possible from the image and uses max-pooling to get only the most important values from the matrix. Such a

high-dimensional matrix (in the case of our full model the output of the contracting path is a tensor of [31, 28, 384]) full of features describing the input sequence is passed to the expansive path which tries to find the correlation between the features passed and the prediction output wanted.

The expansive path building block is very similar to the contracting block with a single difference: convolution-dropout-convolution followed by transpose convolution instead of max-pool operation. The transpose convolutions act as up-sampling layers, the only difference between them and the up-sampling layers is that they contain weights and biases to be fit during the training process. The Conv2DTranpose layers learn the proper kernels to use to achieve the best possible up-sampling capabilities to reach the target image while getting the feature matrix from the contracting path. Fig. 11 presents how the transpose convolution upsamples an example input image matrix.

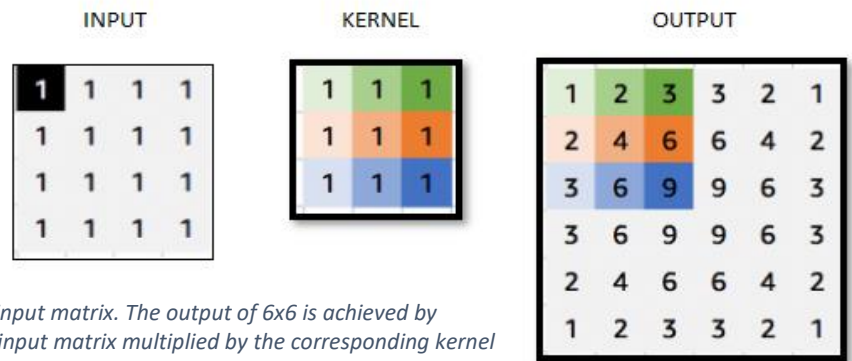


Figure 11 A 3x3 unit kernel applied to 4x4 input matrix. The output of 6x6 is achieved by distributing the value of the pixel from the input matrix multiplied by the corresponding kernel values over the kernel size window on the output matrix. The stride in the example is set to 1 for the simplicity.

The second major difference is that as the expansive path progresses, the number of kernels in the convolutional layer decreases to finally reach the 48 dimensions in the output tensor equal to [496, 448, 48]. The blocks from the same ‘level of depth’ on the contraction part have the numbers of kernels, and usually other parameters as well, equal to the number of kernels in inverted order from the layers in the respective expansive block: i.e. the first contracting block in our full-model from Fig. 12 has the number of kernels equal to 48, 96, 96, 96 while the last expansive block has 96, 96, 96, 48, and similarly the last contracting block has 192, 384, 384, 384 kernels while the first expansive block has 384, 384, 384, 192. What is more, the blocks from those two paths are connected directly in U-Net by using the concatenation of the output of the respective blocks. The same level concatenations work as skip connections and allow for not only a better performance of the model by combating the vanishing gradient but also getting more emphasis on different-level features increasing the precision of the model decoding part.





One of the issues seen with using the U-Net architecture is that the original model was developed by using square-resolution data of 400 by 400 pixels. The Traffic4Cast data initial resolution of 495 by 436 is not compatible with the architecture due to shape mismatch issues when concatenating the output of a contracting block with the respective output of the expansive block. To fix this issue, we have found that the nearest shape not causing the mismatch is equal to 496 by 448 resolution. That is why we have added the Zero-Padding layer to the beginning of the model, and the padding added is (1,0), (6,6) which adds a row of zeroes at the top of the image and six columns of zeroes both at the left and right side of the matrix image. This causes the prediction output of the model to be of the same shape but is easily dealt with by cropping the unnecessary rows to achieve the original shape of 495 by 436.

While the final model was used to produce the submission for the challenge, we created several smaller U-Net models that used the earlier described heavily reduced and pre-processed data. We scale the parameters of the smaller sized models by having the numbers of kernels in each layer as a parameter that doubles in each next contracting path and decreases by half in each next expansive path. Such parameter is tested in different versions: by initially doubling the parameter, making it smaller by half and leaving it as the standard version. We have also tested how the number of random 1-hour sequences taken from the data changes the performance of the model by randomly taking 15 and 30 sequences from every day of the data given.

### 2.2.2 Graph Convolutional Neural Network

The Graph CNN we are using bases on Graph-ResNet [12] and it is a very different deep learning architecture compared to the U-Net. Instead of directly working on an image, a graph representation of the crucial information for the task is input into the model. A graph is a structure that consists of nodes storing some information, edges connecting the nodes, and weights for both the edges and nodes. Every node can be connected with up to eight other nodes which are the neighbouring street-pixels. The connections are stored in an adjacency matrix where elements  $a_{ij}$  are equal to 1 if there is a connection between  $i$  and  $j$  or 0 if there is no connection between them. The convolutions applied in the Graph models are very different from the regular 2D Convolutional layers because of the non-Euclidean space of graphs compared to working on image matrices. The graph convolutional theory provides the Laplacian operator [13]:

$$\Delta = D - A \quad (1)$$



where  $D$  is an  $N$  by  $N$  matrix with the diagonal values equal to the number of connections of each node while the non-diagonal values are equal to 0, this is the Degree matrix.  $A$  is the adjacency matrix described earlier. Using it we can get the vector of Fourier Basis Modes  $\Phi$ , that is used in the final equation of convolution in the spectral domain:

$$g_l = \sum_{l'=1}^p \Phi \widehat{W}_{l,l'} \Phi^T f_{l'} \quad (2)$$

where  $f_l$  equals to the  $l$ -th sequence of features with  $g_l$  as the  $l$ -th feature map and  $\widehat{W}_{l,l'}$  is the  $N$  by  $N$  diagonal matrix of graph spectral filter (trainable parameters).

Figure 13 shows the Graph CNN architecture basis model we have used. The standard Graph Convolution Block is repeated  $n$  times depending on the architecture tested and consists of a sequence of Convolution->Batch Normalisation->Dropout to extract the matrix of crucial features from the data, that then is concatenated with the input to the block followed by Convolution. This forms a skip connection known from ResNet architecture [14], that combats the vanishing gradient and overfitting, and allows for a better performance by putting more emphasis on different level features (pre- and post-block). During the research, we have tested three different types of convolution: Chebyshev, Generalised Graph Convolution and Hypergraph Convolution.

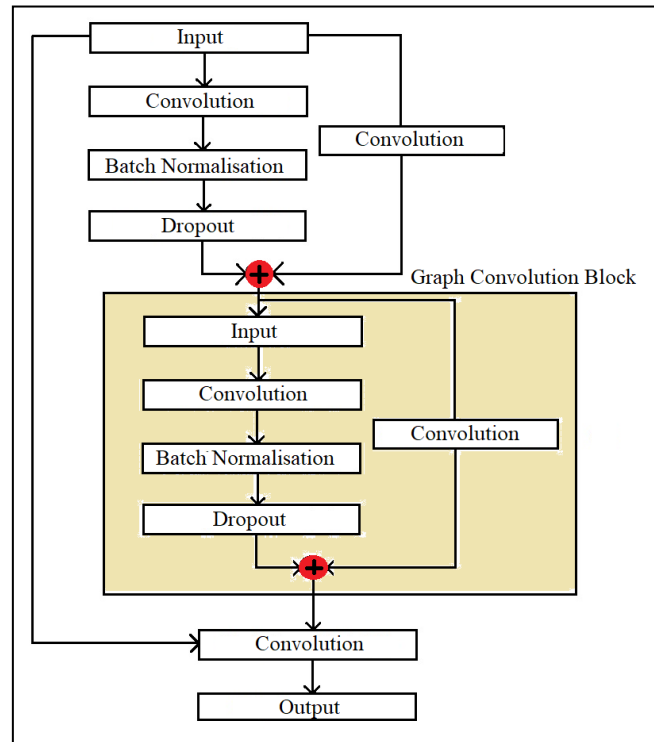


Figure 13 The Graph CNN architecture. The input to the block of convolution->batch norm.->convolution is summed with the output of the block creating a skip connection known from the ResNet architecture. The Graph Convolution Block (beige colour box) is repeated several times (5 or 7) depending on the different architectures used in our testing.

				Parameters		
Model	Convolution Type	No. of Blocks (N)	Layers in Block	Input Block	Blocks 2 to N	Output Layer
Base	Chebyshev	5	ChebConv	(96, 80, K=4)	(80, 80, K=4)	ChebConv(176, 48, K=2)
			BatchNorm	(80, eps=1e-05, momentum=0.1, affine=True)		
			DropOut	Probability = 0.5		
			Skip (ChebConv)	(96, 80, K=1)	(80, 80, K=1)	
			Other parameters of ChebConv: normalization='sym', bias=True			
GENConv	Generalised Graph	7	GENConv	(96, 80)	(80, 80)	GENConv(176, 48)
			BatchNorm	(80, eps=1e-05, momentum=0.1, affine=True)		
			DropOut	Probability = 0.5		
			Skip (GENConv)	(96, 80)	(80, 80)	
			Other parameters of GENConv: aggr: str = 'softmax', p: float = 1.0, eps: float = 1e-07			
HyperConv	Hypergraph	7	HypergraphConv	(96, 80)	(80, 80)	HypergraphConv(176, 48)
			BatchNorm	(80, eps=1e-05, momentum=0.1, affine=True)		
			DropOut	Probability = 0.5		
			Skip (HypergraphConv)	(96, 80)	(80, 80)	
			Other parameters of HypergraphConv: heads=1, concat=True, negative_slope=0.2, dropout=0			
Base-Extended	Chebyshev	7	ChebConv	(96, 90, K=4)	(90, 90, K=4)	ChebConv(186, 48, K=2)
			BatchNorm	(90, eps=1e-05, momentum=0.1, affine=True)		
			DropOut	Probability = 0.4		
			Skip (ChebConv)	(96,90, K=1)	(90, 90, K=1)	
			Other parameters of ChebConv: normalization='sym', bias=True			

Table 1 Different variations of the Graph CNN architectures tested. They mostly differ in the type of convolutional layers used and the number of parameters in those layers. All architectures except the base one have 7 Graph Convolution Blocks.

Table 1 shows the different architectures we have used to solve the traffic prediction task. The number of Graph Convolution Blocks is either 5 or 7, and the crucial change between the models is using the three different types of convolutions. The input to the convolutions in the blocks is equal to 96 (8 channels multiplied by 12 5-minute timesteps) while the number of kernels used is 80. The final output layer is a convolution with 176 input (96 previous input summed by skip connection to 80 output from the block) and the output is equal to 48 (8 channels for 6 5-minute timesteps). The exception from this is the Base-Extended model where we made the number of kernels higher by 10 in the convolutional layers, hence the output layer takes an input of 186 instead of 176. The batch normalisation and dropout parameters are the same between the models to achieve fair comparison with the exception of the Base-Extended model where the dropout probability was lowered to 0.4. Different types of convolutional layers take different input parameters (for example K in ChebConv, slope in Hypergraph) so the inner working of the convolutions need to be explained.

Spectral convolutions use a kernel that is used to multiply the signal, in the case of graph learning it is the feature matrices of nodes. Chebyshev Convolution uses a kernel consisting of Chebyshev polynomials of the Laplacian eigenvalues from a diagonal matrix [15]. The kernel is described as:

$$g_{\theta}(\Lambda) = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda}) \quad (3)$$

where  $g_{\theta}$  is the kernel of Chebyshev coefficients  $\theta$  applied to Laplacian eigenvalues  $\Lambda$  ( $\tilde{\Lambda}$  is the scaled),  $k$  is the smallest order neighbourhood,  $K$  is the largest order neighbourhood,  $T$  is the  $k$ -th order Chebyshev polynomials.

In other words, the kernel applied is equal to the sum of Chebyshev polynomial kernels multiplied by the scaled Laplacian eigenvalues matrix for orders from  $k=0$  to  $K-1$ . The  $K$  parameter that we set to 4 for in-block convolutions and 1 for skip connection determines the order considered within the convolution, where 1<sup>st</sup> order uses similarity metric between 2 nodes based on the immediate neighbourhood, 2<sup>nd</sup> order takes into consideration further neighbourhood structures, and so on. The higher number of  $K$  increases the depth of considered nodes during the similarity calculation. The high number of  $K$  within the Graph Convolution Block in our implementation is used to get the temporal correlation within a large area, while the  $K$  equal to 1 is set for skip connection since we want only compare and then sum the input and output from the block.

Hypergraph Convolution uses hypergraphs instead of regular graphs [16]. In a normal graph, each edge connects 2 nodes only, while the hypergraph implementation instead of edges uses small, local clusters connecting several nodes at the same time. This approach allows to better combine small areas and therefore better learn local temporal dynamics of traffic flow in our case. In both Hypergraph Convolution and Chebyshev Convolution, we use symmetric normalisation to prevent eigenvalues of the operators exceed 1, which also prevents exploding and vanishing gradients. The issue of time constraints has not allowed us to prepare the proper input graph data in order to take the advantage of hypergraph convolution therefore the tests made were done on normal graphs to empirically see the results of such an approach.

Generalized Convolution is a convolutional layer suggested by Li et al in 2020 [17] to make the stacking of many layers in very deep architectures more efficient by combating the vanishing gradient, over-smoothing or over-fitting issues found in very deep networks. The

structure focuses on aggregation functions, the functions that take as input a graph  $G$  and output a transformed graph  $G'$ . The two supported by PyTorch implementation of GENConv aggregation functions are PowerMean and SoftMax which is a generalised mean-max aggregation function that used in our project due to the emphasis on large important values (i.e. high volumes on traffic junctions are key to predict the next timestep's flow).

The static data explained previously gives the information about the road network and its connectivity, so the number of nodes (i.e. pixels in the road network) and edges (the connections between the pixels) is easily calculated resulting in the graph structure. However, it is the dynamic data that has the volume and speed information and that must be used during the training. The algorithm to use the dynamic and static data together to create the proper training data works as described below:

1. The graph structure (nodes and edges) is created using static data which is the road network of a city with different raster resolutions.
2. The nodes in the graphs have the pixel coordinates saved as their IDs.
3. The coordinates are compared with the training image data.
4. The volume and speed information is then extracted and used for the training process.
5. The prediction output for tests and the submission is created by inverting the process (which is taking the node IDs – coordinates – and using them to create the 495x436x8 image prediction).

### 2.2.3 Naïve and Weighted Averages

As a simple and straightforward baseline approach that we could quickly compare the prediction results of U-Net CNN and Graph CNN, we have created a simple naïve and weighted averages models. This statistical model uses as an input a tensor of the same length sequence that the deep learning models do which is 12 5-minute time steps of data and then calculates the mean of the volume and speeds channels over the one hour period. The output is treated as the prediction of the traffic flow during the next 5-minute timestep. Ideally, we would like to take a different approach of averaging over a certain time in a day for the full data (i.e. take the 180 days of data, and average the traffic flow at 9:00 am in all of the days and so on). This would have definitely resulted in better accuracy of the prediction, however, such an approach was not possible due to the testing and submission data not having any timestamps in the 2021 Traffic4Cast edition, unlike in the previous years where the times of the testing input sequences were provided.

It is easy to estimate that the ground truth traffic flow 5 minutes after the one hour input will most likely be more similar to the last timesteps in the sequence – so closer in time to the prediction itself, than the data from the beginning of the sequence which has almost an hour difference in time from the prediction. That is why we use weighted averaging over the sequence, putting more weight into the data closer to the prediction in time. The average model used in the research has three variations of weights:

- a) naïve average: weights equal to [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] for each of the 12 timesteps in the input sequence which gives all data the same importance;
- b) weights flat: [1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 5, 6] with a moderate increase in weights the closer the sequence data is to the prediction;
- c) weights steep: [1, 1, 1, 1, 1, 2, 2, 3, 3, 5, 7, 9] these weights dramatically increase the importance of the closest time-steps to the actual prediction;

Similarly to how we described the generation of the 45 and 60-minute prediction for U-Net and Graph Net earlier, we make the average model predictions of later time-frames by taking the previous prediction and adding the necessary amount of previous input to form a new one-hour input sequence: for predicting the next 15-minute timestep, we take the output of 5-minute and 10-minute prediction and add them to the last 50 minutes of the original input.

#### 2.2.4 Optimizer and Hyperparameters.

The project uses AdaBelief optimiser which is a recently released improvement upon the standard Adam optimiser [18]. The optimisation made by AdaBelief can be described as:

*while  $\theta$  not converged:*

$$t = t + 1 \quad (4)$$

$$g_t = \nabla_{\theta} f_t(\theta_{t-1}) \quad (5)$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \quad (6)$$

$$s_t = \beta_2 s_{t-1} + (1 - \beta_2) (g_t - m_t)^2 \quad (7)$$

*bias correction:*

$$\widehat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad \widehat{s}_t = \frac{s_t + \epsilon}{1 - \beta_2^t} \quad (8)$$

*update:*

$$\theta_t = \Pi F, \sqrt{\widehat{s}_t} (\theta_{t-1} - \frac{\alpha m_t}{\sqrt{\widehat{s}_t + \epsilon}}) \quad (9)$$

where  $f(\theta)$  a loss function to optimise weights  $\theta$ ,  $g_t$  is the gradient at timestep  $t$ ,  $m_t$  is its exponential moving average,  $\beta_1$  and  $\beta_2$  are the hyperparameters set by us (by default: 0.9 and 0.999 respectively),  $\alpha$  is the learning rate and  $\epsilon$  denotes a very small number to avoid zero denominator.

While AdaBelief optimiser is almost identical to Adam, there is one change made during the calculation of the EMA of squared gradient: it has the multiplication by  $(g_t - m_t)^2$  which is the square of the difference between the gradient and its exponential moving average. It is a crucial change allowing AdaBelief to take larger steps when the gradient is small and small steps during the times when the gradient is large which is further explained using Fig. 14.

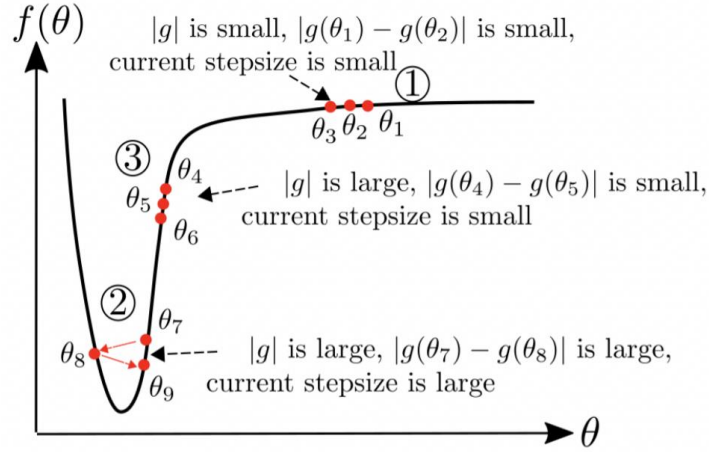


Figure 14 Plot of Loss error used to show the differences between AdaBelief and Adam behaviour [18].

A perfect optimiser would take a large step in region 1 due to the small gradient, and both Adam and AdaBelief do exactly that. Region 2 has a large gradient so the perfect action would be again taking a small step, which is done again by both of the optimisers. The difference is in region 3 where only AdaBelief takes a large step due to its small denominator caused by a small difference in  $(g_t - m_t)$  which makes the  $s_t$  value also low. This small difference makes AdaBelief converge faster and more easily find the global minima of the loss function.

The most often used activation function in our architectures is the ReLU activation function shown in Fig. 15, due to its many advantages such as being designed to avoid the vanishing gradient problem, fast convergence and ease of optimisation due to its simplicity [19].

$$\text{Rectified Linear Units (ReLU):} \quad f(x) = \max(0, x) \quad (10)$$

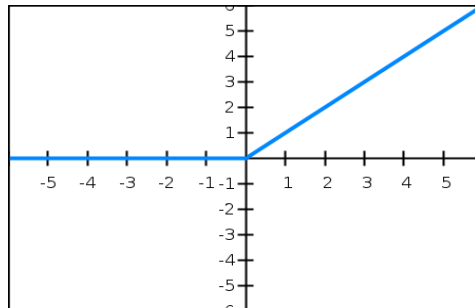


Figure 15 The ReLU activation function that is used in most layers in our architectures.

Other training parameters include the batch size of 2 dynamic data files (so 18 times 2 input sequences) for U-Net and the size of 3 for Graph CNN. The models were trained for 5 epochs for U-Net and 1 epoch for Graph CNN, during those the models always seemed to successfully converge. The learning rate for U-Net and GCNN were  $1 \times 10^{-5}$  and  $1 \times 10^{-4}$  respectively. We use the reduction of learning rate when there is no improvement for 2 consecutive epochs during the U-Net training. Since the problem of traffic flow in our case is a regression problem, the loss function used is the Mean Squared Error:

$$MSE = \frac{\sum_{i=1}^n (y_i - \bar{y}_i)^2}{n}$$

where  $y_i - \bar{y}_i$  is the difference between the predicted and target values.

## CHAPTER III: RESULTS AND DISCUSSION

While the aim of this project was to minimize the error loss score for the submission to the Traffic4Cast challenge, we have produced a lot of results on our local test dataset in order to properly evaluate the models and different approaches, and hence find the best ones to submit to the competition. The results subsection presents the findings in a visual form with a brief description of what each figure presents, while the in-depth analysis about possible causes of the differences in performances of the models can be found in the discussions subsection.

### 3.1 Results

The reduced-size U-Nets architectures were trained on Antwerp training data with 180 pre-covid and 164 post-covid days of data. The last 16 post-covid data files were used for evaluation purposes to see the performance of different models' configurations on new, unseen data.



Figure 16 The MSE error for 6 different models of U-Net. The full data (blue) is the pre-covid and post-covid data combined (344 days) while the half data (orange) denotes the models that were trained only on post-covid data. Standard kernel size means the first convolutional layer of U-Net uses 48 kernels, while double has 96 kernels.

Having the results which ensured us on the final model parameters we have trained full U-Net models with no data downscaling or averaging the channels. Four models were trained, each on a different city from the core competition: Berlin, Chicago, Istanbul and Melbourne.



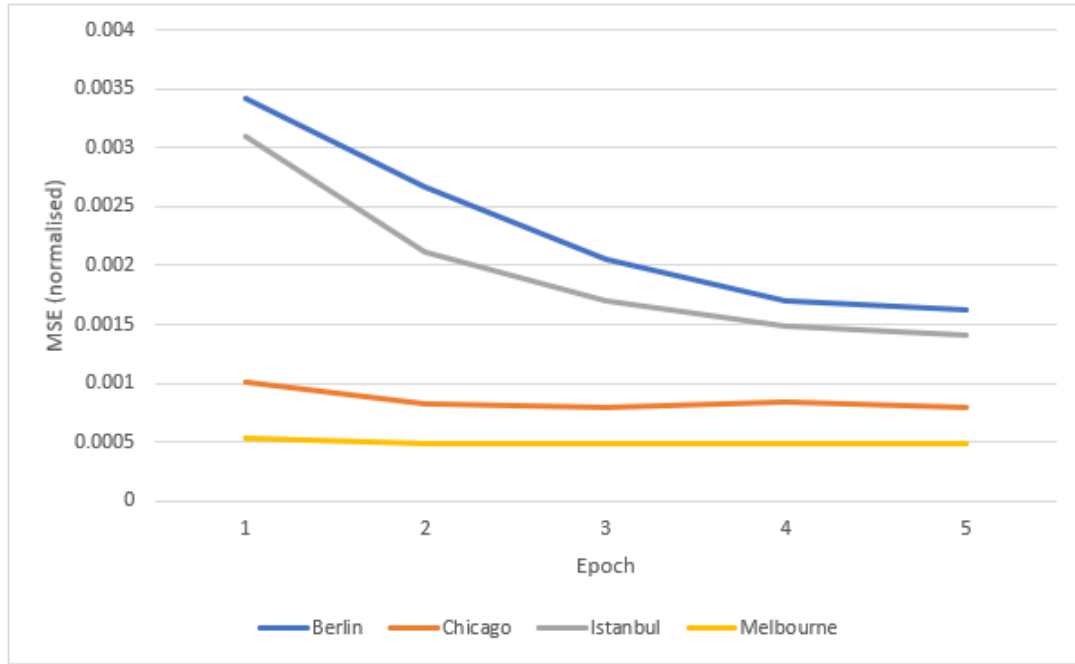


Figure 17 The training MSE error for the final models trained on the full sized data of the four cities from the core competition.

We wanted to explore the effect of reducing the image pixel-information by a substantial amount (from 495x436 to 200x200 and from 8 channels to only 2) hence we compared the MSE performance of the full model with the reduced model on the testing set.

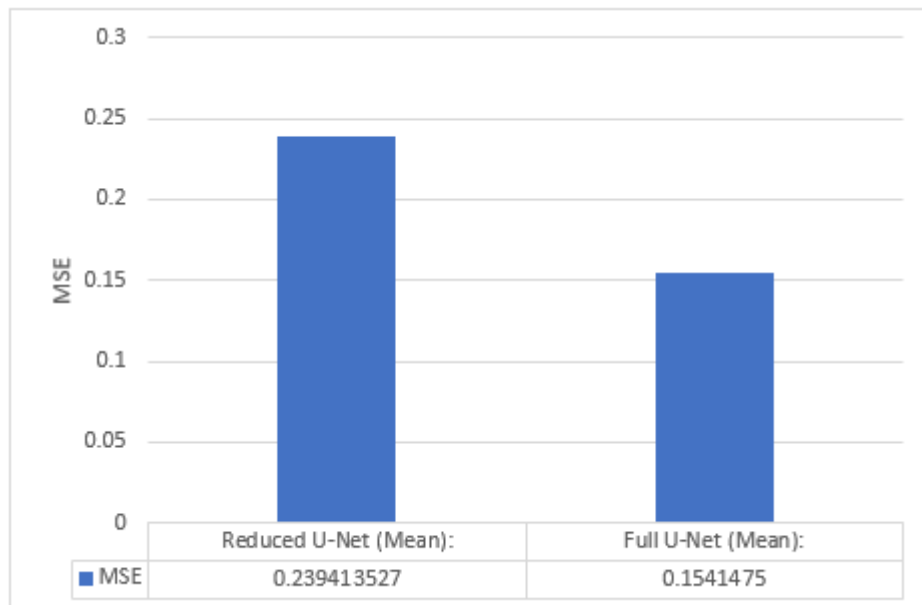


Figure 18 The mean MSE for the reduced-data U-Net and full sized U-Net. The models were trained on the core cities using all data days, each model used the best hyperparameters found empirically in the previous step (Fig. 16).

To check the generalization capabilities of the final U-Net models, we evaluated them on the unseen cities data of Bangkok and Barcelona.

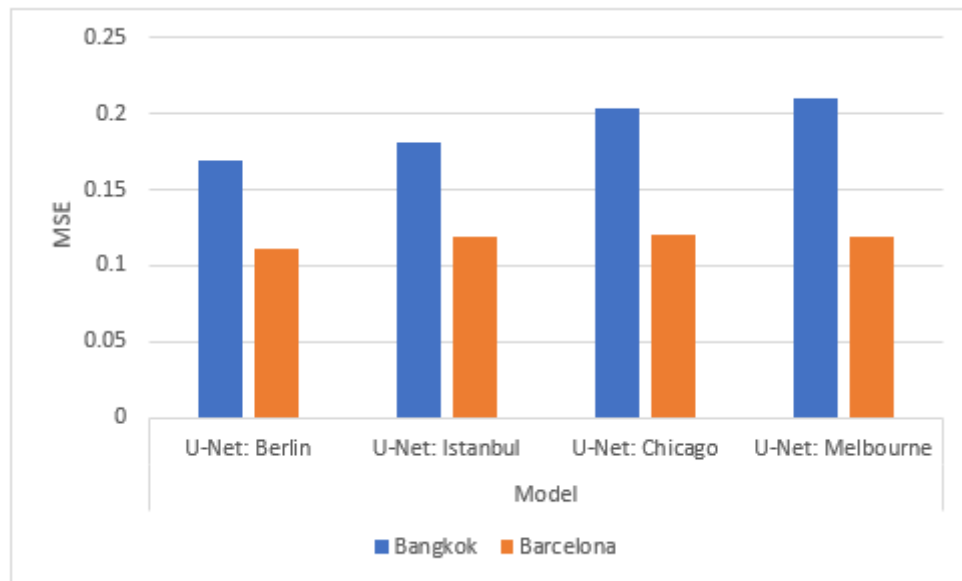


Figure 19 Mean MSE achieved by the final U-Net models during the evaluation on new cities.

The same data was used to make a comparison to evaluate how well the U-Net performs compared to different average models.

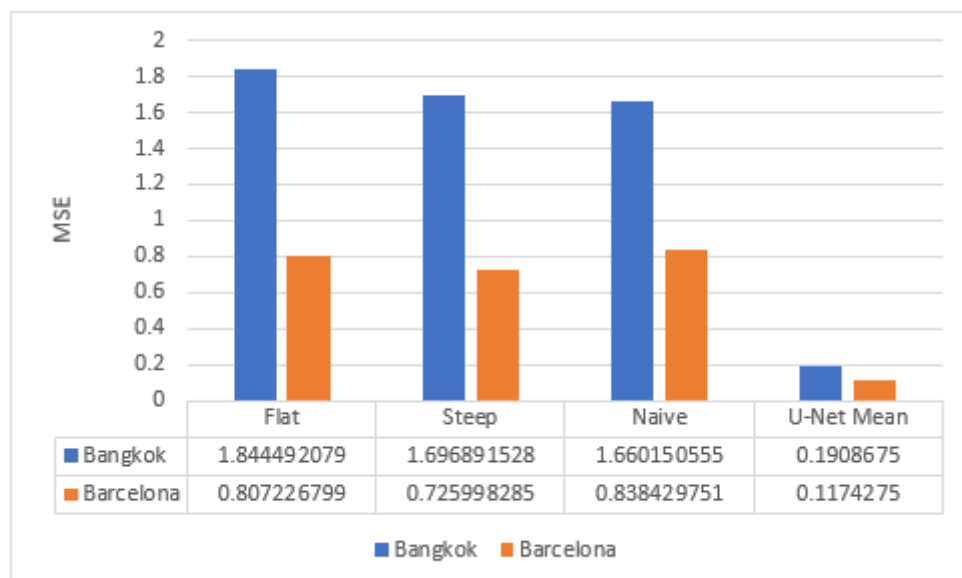


Figure 20 Mean MSE achieved by the average models compared to the mean MSE of the final U-Net models on Bangkok and Barcelona cities.

The Graph CNN had 4 different models trained: Base (Chebyshev Convolution), Hypergraph, GENConv (Generalised Graph Convolution), and Base-Extended (Base model with 2 more blocks and a larger number of kernels) as described earlier. The training was done on half data for Berlin city due to time and resources constraints.

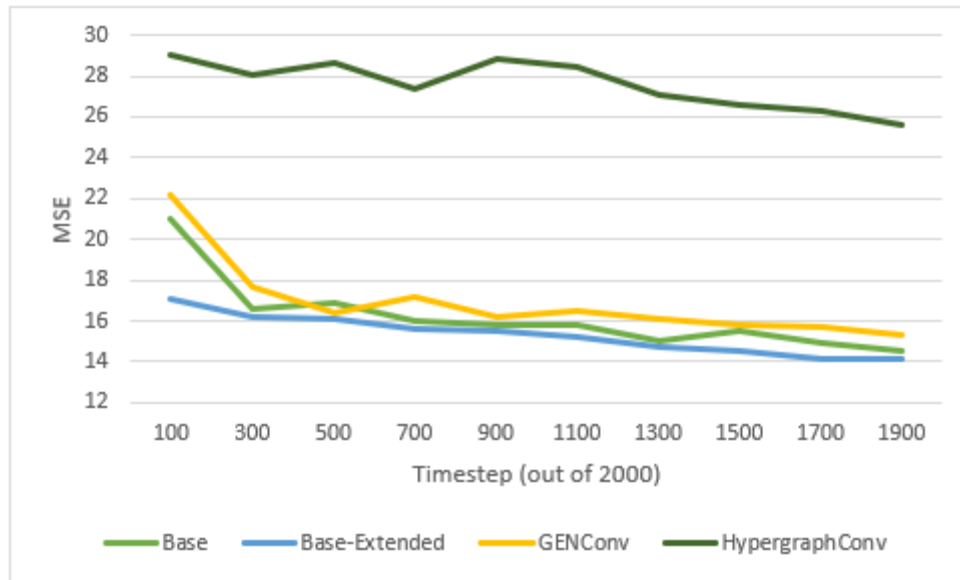


Figure 21 The training MSE error for the Graph CNN models trained on the half-data of Berlin.

Additionally, we have tested how different is the time for AdaBelief and Adam optimisers to converge.

	Convergence (GraphNet Base-Extended)	
	Timesteps	Minutes
<b>Adam</b>	1850	60
<b>AdaBelief</b>	1200	45

Table 2 The difference in timesteps and time between Adam and AdaBelief to find the converging point.

All the final models (four Graph CNN and the four full U-Net models) were used to predict the 5, 10, 15, 30, 45 and 60-minute traffic flow into the future given the testing sequences from the test submission data. There is one file per core city, each with 100 hours of testing sequences. The submissions were sent to the Traffic4Cast challenge where they were evaluated and shown on the leaderboard.




Pos.	Name	Team/User	Score
1	iarai_unet_baseline_sep	IARAI-BASELINES - Moritz Neun	51.282680678368
2	iarai_gcn_one_epoch_BERLIN	christian.eichenberger@iarai.ac.at - Christian Eichenberger	51.714257740974
3	testgcn	oliwiech	51.891605806351 
4	Test GCN 3	UoN sub3 - panas41298	51.930465149879
5	sub_6	polymathAB - polymathab16	52.001782894135
6	gcn_long	oliwiech	52.13072142601 
7	IARAI_henry_gcn_test	iarai_henry - Henry Martin	52.202835798264
8	gcn_base_extended	oliwiech	52.205313277245 
9	sub_7	polymathAB - polymathab16	52.51177983284
10	Alchera_baseline	jaysantokhi - jaysantokhi	53.280712032318

Figure 22 The top 10 of the Traffic4Cast 2021 challenge leaderboard. The score is the sum of all the MSE of the predictions from the 400 testing hours (100 test hours per core city). Our models are shown on 3<sup>rd</sup>, 4<sup>th</sup>, 6<sup>th</sup> and 8<sup>th</sup> place and they correspond to: Base GCNN with Adam, Base GCNN with Adabelief, GENConv GCNN and Base-Extended models respectively. The Hypergraph and U-Net models achieved score outside the top 10.

Additionally, we have made a submission to the extended challenge where the Base-Extended model had to make a prediction on two new, unseen cities of Vienna and New York.


Pos.	Name	Team/User	Score
1	iarai_gcn_one_epoch_BERLIN	Christian Eichenberger	61.461115264893
2	gcn_base_extended_spatiotemporal	oliwiech	61.505285453796 
3	sub_1_ext	polymathAB - polymathab16	62.227761459351

Figure 23 The top scores of the Traffic4Cast extended challenge leaderboard, state at 10<sup>th</sup> of August 2021.

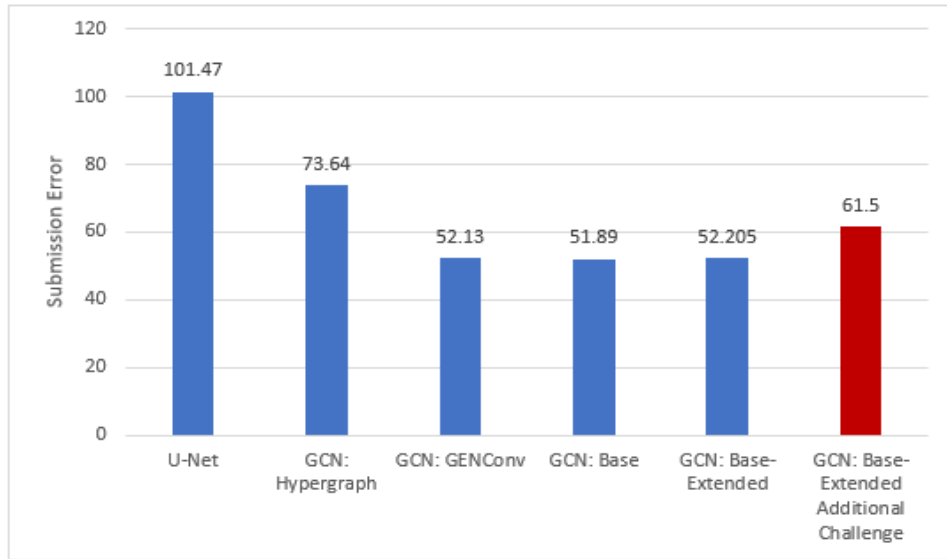


Figure 24 The plot of MSE (submission) score from the Traffic4Cast 2021 leaderboard achieved by all of our final models sent to both core and extended challenge..

To check visually how well the models are performing, we plot the example predictions made by the best models of Graph CNN, U-Net and Average model, together with the last 20-minute data of the input sequence. For ease of understanding and space limitation, we visualize the average of the 4 directions channels for both speed and volume.

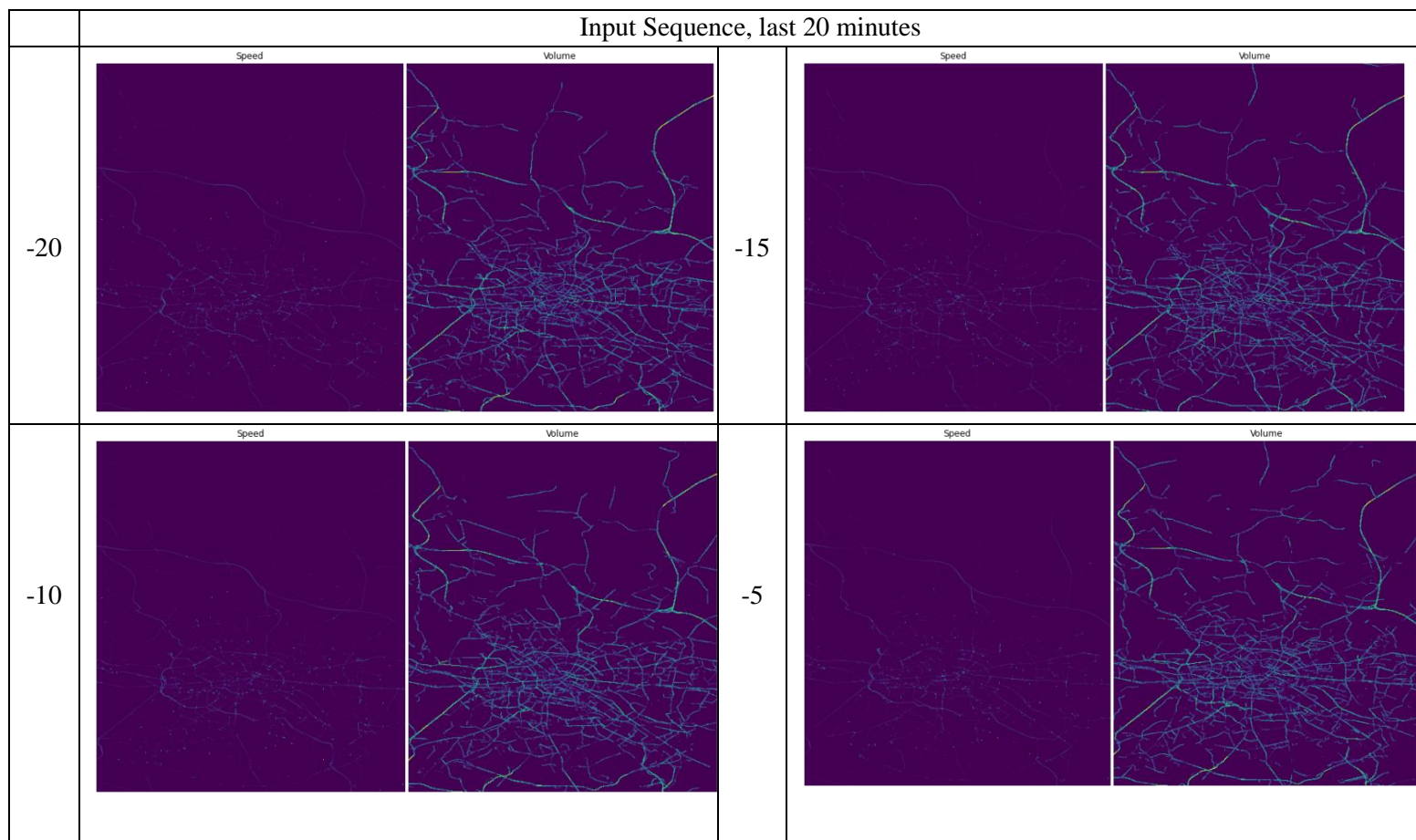


Figure 25 The visualisation of the last 20 minutes of the 1 hour input sequence.



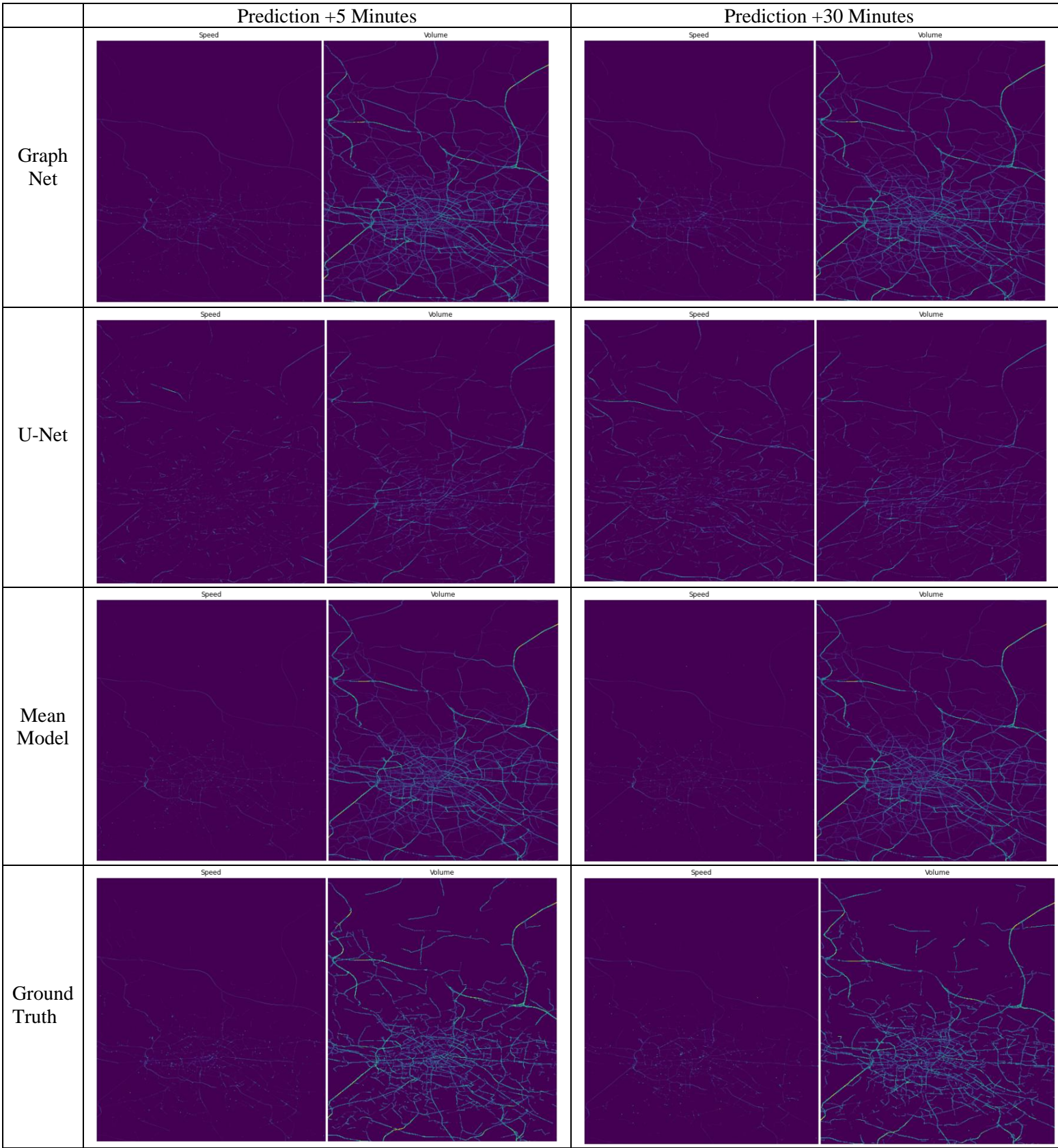


Figure 26 The visualisation of the predictions (5 minutes and 30 minutes into the future) made by Graph Net, U-Net, Average Model, and the corresponding Ground Truth.

### 3.2 Discussion

The first results presented in Figure 16 shows how important the number of kernel filters and the amount of the training data really is to let the models properly capture the important features in order to generate successful predictions. Furthermore, the major aim of researching these particular results was to get the answer to whether using the additional pre-covid data is beneficial to the task since the test data is post-covid only. As we can see, the models train on half data, so only post-covid, achieve better results in the testing evaluation than the models trained on full data. It can be explained by the fact that the pre-covid and post-covid data are very different, especially in the volume, therefore adding the extra pre-covid data instead of improving the results introduces ‘noise’ of different traffic than the post-covid one. As expected, the higher number of kernels lowered the error due to having more depth allowing to better capture the lower-level features of the traffic flow. Additionally, increasing the number of sequences per day from 15 to 30 did not result in any major difference in the results, hence the 15 sequences per day are preferred due to the much lower time required for training.

The training of full U-Net models, Fig. 17, shows that the models trained on Berlin and Istanbul data result in the largest improvement in the MSE over the training time. This is most likely caused that the road network in those two cities is much larger and more complex than the Chicago and Melbourne networks which converged almost immediately. This was the main reason to pick Berlin as the final training city for the Graph CNN models – the convoluted road networks allow for better learning of various patterns and therefore better generalisation properties.

This choice is even further proven in Fig. 19 where the U-Net models trained on different cities are used to predict test data on new, unseen cities of Bangkok and Barcelona. The Berlin model has significantly better results and generalises better than the other 3 models. The MSE score difference between Bangkok and Barcelona can be explained by the length of the road network which is equal to 5 511 799 pixels for Bangkok and 2 660 217 pixels for Barcelona. The larger road network produces more room for a mistaken prediction and makes the learning harder – the number of features is significantly higher since the volume and speed is distributed on the twice larger road network.

Going back to Fig. 18 which shows the difference in MSE between reduced U-Net (mean: 0.239) and full U-Net (mean: 0.154) and proves that a lot of information was lost by reducing the image dimensions and averaging the 4 volume and 4 speed channels into 1 each. This

reduction made each pixel give information for 200x200 square meters instead of the original 100x100, resulting in significant information reduction about the traffic propagation and not being able to capture detailed changes in the traffic flow.

Figure 20 which shows the comparison between U-Net results and different Average Models (Flat, Steep and Naïve) proves that the deep learning approach is significantly (8 times) better than the traditional statistical model. The result that the steep weights in the Average Model are the best is the intuitive and most obvious finding. It is easy to deduct that the next 5-minute prediction is much more similar to the last timesteps from the input sequence that are very close in time to the actual prediction, than to the almost one-hour old data from the beginning of the input sequence.

The graph training showed in Fig. 21 confirms that the Base model, Base-Extended and GENConv achieve very similar results but HypergraphConv gets a much higher error. This is due to the fact that the Hypergraph Convolution works best when the input data is specifically prepared to be a proper hypergraph that clusters the nodes instead of the current regular graph where the edges connect only 2 nodes. The time constraint, and also the other results that achieved a very good score made us decide not to pursue the implementation of hypergraph data representation.

Additionally, Table 2, shows that the AdaBelief achieves faster convergence than the Adam optimiser. The difference in time is around 15 minutes and 650 timesteps (out of 2000) to achieve a converging point. However, the MSE submission score was very similar between the two.

Figures 22, 23 and 24 that show the submission results of all our main models (4 Graph CNNs and the U-Net models) prove that the Graph Convolutional Neural Network is superior and achieves significantly better results than the U-Net in the task of traffic flow prediction, and provides better properties in capturing the temporal dynamics and generalising to other, newly seen data. Surprisingly, adding more layers to the base models and increasing the number of kernels did not improve its score compared to the base model. This can be explained by the likely overfitting – the deep model learned more about the representation of the training set and therefore generalised worse on the submission data. The GENConv model and Base model generally achieved very similar scores with a difference of 0.24 MSE between them. U-Nets and Hypergraph GCNN results were significantly lower – the differences between them and the best Base GCNN model were: 49.58 and 21.75 MSE respectively. The Base-Extended



model achieved 2<sup>nd</sup> score in the extended challenge showing its good generalisation properties to newly introduced cities of Vienna and New York.

The models scored highly in the leaderboard for both core and extended cities beating many competitors with more resources. We can deduct that most likely using full data during training of the Graph CNN instead of only half data for one city would improve the score even further.

The visualisations shown in Fig. 26, show that the Steep and Graph CNN produces the best output for 5-minute prediction. The U-Net however struggles to capture the intensity of the volumes and speeds, and does not detect the shape of the road networks very well. In the 30-minute prediction, however, it seems to produce closer to the Ground Truth road network than the Steep model. However, the Graph CNN achieves results that look significantly better than the other models capturing very well both the shape of the network and the intensities of speed and volumes in both 5-minute and 30-minute timeframes.

## CHAPTER IV: CONCLUSION

The main purpose of the research was to analyse the current state of the art deep learning approaches for traffic flow prediction task and advance their performance in order to achieve a good score on the Traffic4Cast challenge hosted by the Institute of Advanced Research in Artificial Intelligence.

The traffic flow prediction is an important problem, especially nowadays where so many individuals and businesses use land vehicle transport since high congestion can and improper traffic management heavily impacts the economy and fuel environmental damage. Advancing the current traffic flow prediction knowledge about the methods and their performances is crucial to combat global warming as well as to help businesses and governments reduce risks and costs of logistics.

Our models, trained on the data was provided by IARAI with collaboration with HERE Technologies that consists of four cities: Berlin, Istanbul, Chicago and Melbourne achieve results in predicting the next one hour of traffic flow of top-level quality which is proven by the Traffic4Cast 2021 challenge leaderboard.

The Graph Convolutional Neural Network is a significantly better approach than traditional statistical methods such as weighted averaging or more basic Convolutional Neural Networks architectures like U-Net. The basic model that uses Chebyshev Convolution produces very similar to the ground-truth visualisations of the next hour predicted of the traffic flow, its speeds and volumes for the four different directions.

The experiments that we have conducted that include using different convolutions: Hypergraph and Generalized Graph Convolution, as well as increasing the complexity of the model by making the number of layers and kernels in the model higher showed that there was no improvement in the submission scored compared to the base Chebyshev Convolution model. Furthermore, another finding is that the AdaBelief significantly reduces the training time needed to converge while not producing worse results and is the preferred optimiser compared to Adam optimiser.

There are many ideas for further improvement on this work. The major one is training the Base Graph CNN on full data instead of only half of the data for one city. Despite the very limited data and making the model generalize the other three cities, the Base model achieved a very good score on the leaderboard. Training it on full data for all the cities is likely going to improve

the submission score even further. Another possible idea for improvement is to use the Graph Pooling layers from PyTorch Geometric library which would allow for even deeper and larger models without meeting the overfitting issue. A deeper model combining the U-Net and the Graph CNN model could also improve the prediction capabilities of the architecture by for example feeding the output of the Graph CNN into a U-Net which would learn the relation between the Graph CNN prediction and the Ground Truth directly. The last suggestion would be to fine-tune the hyperparameters using grid search with cross-validation. This approach would find the best possible hyperparameters but requires large GPU power to have short enough training times to check different possible combinations of the parameters.

The major issue met during the research conducted was the very large amount of data provided combined with the lack of stable resources to efficiently train different deep learning architectures. Despite that, the aim of this project was met successfully with very good results that are higher than most competitors' scores. The deep learning models, especially the Graph CNN, could be capably and reliably used in real-life Intelligent Transportation Systems in order to reduce congestion and help individuals and businesses to manage the traffic in a better way due to the prediction knowledge provided by our models.

## REFERENCES

1. MacKenzie, J. J., & Walsh, M. P. (1990). Driving forces: Motor vehicle trends and their implications for global warming, energy strategies, and transportation planning.
2. Cohn, N. (2020). Traffic data: the key to effective road traffic management. <https://www.tomtom.com/blog/traffic-and-travel-information/road-traffic-management/>
3. Martin, H., Hong, Y., Bucher, D., Rupprecht, C., & Buffat, R. (2019). Traffic4cast-Traffic Map Movie Forecasting--Team MIE-Lab. arXiv preprint arXiv:1910.13824.
4. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, 521(7553), 436-444.
5. Lv, Y., Duan, Y., Kang, W., Li, Z., & Wang, F. Y. (2014). Traffic flow prediction with big data: a deep learning approach. *IEEE Transactions on Intelligent Transportation Systems*, 16(2), 865-873.
6. Eichenberger, C. (2021). Traffic4cast 2021 – Temporal and Spatial Few-Shot Transfer Learning in Traffic Map Movie Forecasting. <https://github.com/iarai/NeurIPS2021-traffic4cast>
7. Neun, M. (2021). Looking into the road graph. Institute of Advanced Research in Artificial Intelligence. <https://www.iarai.ac.at/traffic4cast/forums/topic/looking-into-the-road-graph/> (<https://scrnli.com/3xxB74qiX0w1Sg>)
8. Google Developers. Tensorflow Documentation. [https://www.tensorflow.org/api\\_docs/python/tf/data/Dataset](https://www.tensorflow.org/api_docs/python/tf/data/Dataset)
9. Ronneberger, O., Fischer, P., & Brox, T. (2015, October). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention* (pp. 234-241). Springer, Cham.
10. O'Shea, K., & Nash, R. (2015). An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458.
11. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1), 1929-1958.
12. Martin, H., Bucher, D., Hong, Y., Buffat, R., Rupprecht, C., & Raubal, M. (2020, August). Graph-ResNets for short-term traffic forecasts in almost unknown cities. In *NeurIPS 2019 Competition and Demonstration Track* (pp. 153-163). PMLR.
13. Bresson, X. (2017). Convolutional neural networks on graphs. *Advances in neural information processing systems*, 29, 3844-3852.
14. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).
15. Defferrard, M., Bresson, X., & Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 3844-3852.

16. Bai, S., Zhang, F., & Torr, P. H. (2021). Hypergraph convolution and hypergraph attention. *Pattern Recognition*, 110, 107637.
17. Li, G., Xiong, C., Thabet, A., & Ghanem, B. (2020). Deepergcnn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*.
18. Zhuang, J., Tang, T., Ding, Y., Tatikonda, S., Dvornek, N., Papademetris, X., & Duncan, J. S. (2020). Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. *arXiv preprint arXiv:2010.07468*.
19. Agarap, A. F. (2018). Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*.